2011

Network flow algorithms for wireless networks and design and analysis of rate compatible LDPC codes

Cuizhu Shi
Iowa State University

Network flow algorithms for wireless networks and design and analysis of rate compatible LDPC codes

by

Cuizhu Shi

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Program of Study Committee:
Aditya Ramamoorthy, Major Professor
Zhengdao Wang
Nicola Elia
Sang W. Kim
Ryan Martin

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to many people who made this thesis possible.

I would like to thank my parents for giving me life and supporting me all the time. I would like to thank my husband and my lovely twins for their endless love.

Special thanks to Prof. Aditya Ramamoorthy for the continuous support of my Ph.D study and research. I am really grateful for his patience, inspiration and enthusiasm all the time. This thesis couldn't have been finished without his support and guidance.

I also wish to thank my committee members, including Prof. Zhengdao Wang, Prof. Nicola Elia, Prof. Sang W. Kim and Prof. Ryan Martin, for their encouragement, efforts and insightful comments.

Many thanks to all friends in Iowa State University for the insightful discussions, the friendship and the amazing four years we had together.

ABSTRACT

While Shannon already characterized the capacity of point-to-point channels back in 1948, characterizing the capacity of wireless networks has been a challenging problem. The deterministic channel model proposed by Avestimehr, etc. (2007 - 1) has been a promising approach for approximating the Gaussian channel capacity and has been widely studied recently. Motivated by this model, an improved combinatorial algorithm is considered for finding the unicast capacity for wireless information flow on such deterministic networks in the first part of this thesis. Our algorithm fully explores the useful combinatorial features intrinsic in the problem. Our improvement applies generally with any size of finite fields associated with the channel model. Comparing with other related algorithms, our improved algorithm has very competitive performance in complexity.

In the second part of our work, we consider the design and analysis of rate-compatible LDPC codes. Rate-compatible LDPC codes are basically a family of nested codes, operating at different code rates and all of them can be encoded and decoded using a single encoder and decoder pair. Those properties make rate-compatible LDPC codes a good choice for changing channel conditions, like in wireless communications. The previous work on the design and analysis of LDPC codes are all targeting at a specific code rate and no work is known on the design and analysis of rate-compatible LDPC codes so that the code performance at all code rates in the family is manageable and predictable. In our work, we proposed algorithms for the design and analysis of rate-compatible LDPC codes with good performance and make the code performance at all code rates manageable and predictable. Our work is based on E^2RC codes, while our approaches in the design and analysis can be applied more generally not only to E^2RC codes, but to other suitable scenarios, like the design of IRA codes. Most encouragingly, we obtain families of rate-compatible codes whose gaps to capacity are at most 0.3 dB across the range of rates when the maximum variable node degree is twenty, which is very promising compared with other existing results.

CHAPTER 1. OVERVIEW

Two fundamental questions in the design of communication systems are: (1) What are the theoretical limits on the amount of information that can be reliably transmitted over given communication systems? (2) What are the practical transmission schemes to approach these limits? Information theory and coding theory have been developed trying to answer these two questions following Shannon's pioneering work in 1948 (Shannon (1948)). Significant advances have been seen since then and bring us closer to the answers. Despite all the progress, there are still many open areas. The work in this thesis contributes to finding answers to these two questions in some open areas.

In this thesis, the capacity of communication systems refers to the theoretical limits on the information rate that can be achieved with arbitrarily small error probability. While Shannon already characterized the capacity of point-to-point channels back in 1948, characterizing the capacity of wireless networks has been a challenging problem for the past few decades. While the exact capacity characterization seems to be a distant goal, there is an increasing research interest in studying the approximate capacity using some simplified channel models. The deterministic channel model proposed by Avestimehr, etc. (2007 - 1) (referred as ADT model thereafter) has been a promising approach for approximating the Gaussian channel capacity and has been widely studied recently.

Motivated by this model, an improved combinatorial algorithm is considered for finding the unicast capacity for wireless information flow on such deterministic networks in the first part of this thesis. Our algorithm fully explores the useful combinatorial features intrinsic in the problem. Our improvement applies generally with any size of finite fields associated with the channel model. Comparing with other algorithms on solving the same problem, our improved algorithm is very competitive in terms of complexity.

The first part of our work contributes to efficiently computing the capacities of given networks, while the second part of our work contributes to designing efficient channel codes approaching the theoretical limits established above.

Channel coding theory can be divided into two branches, classical coding and modern coding (Richardson & Urbanke (2008)). The classical coding is featured by algebraic codes, like linear block codes and convolutional codes, with Reed-Solomon codes as a representative, while the modern coding is featured by efficient iterative probabilistic local decoding and random sparse code graphs, with Turbo codes and low-density parity-check (LDPC) codes as two main representatives.

In the second part of our work, we consider the design and analysis of rate-compatible LDPC codes. Rate-compatible LDPC codes can be described by a mother code and puncturing patterns. They are basically a family of nested codes where coded bits of higher rate codes are embedded in those of lower rate codes. The code corresponding to the lowest rate is called mother code and all other codes can be obtained by puncturing the mother code. The family of codes operates at different code rates and all of them can be encoded and decoded using a single encoder and decoder pair. Those properties make rate-compatible LDPC codes a good choice for changing channel conditions, like in wireless communications. The previous work on the design and analysis of LDPC codes are all targeting at a specific code rate and no work is known on the design and analysis of rate-compatible LDPC codes so that the code performance at all code rates in the family is manageable and predictable. In our work, we proposed algorithms for the design and analysis of rate-compatible LDPC codes with good puncturing performance and make the code performance at all code rates manageable and predictable. Our work is based on E^2RC codes, while our approaches in the design and analysis can be applied more generally not only to E^2RC codes, but to other suitable scenarios, like the design of IRA codes (Jin, etc. (2000)). Our approaches are described in detail in Chapter 4. Most encouragingly, we obtain families of rate-compatible codes whose gaps to capacity are at most 0.3 dB across the range of rates when the maximum variable node degree is twenty, which is very promising compared with other existing results.

Specifically, we consider the design and analysis of the efficiently-encodable rate-compatible (E^2RC) irregular LDPC codes proposed in previous work. In this part we introduce semi-structured E^2RC-like codes and protograph E^2RC codes. EXIT chart based methods are developed for the design of semi-structured E^2RC-like codes that allow us to determine near-optimal degree distributions for the systematic part of the code while taking into account the structure of the deterministic parity part, thus resolving one of the open issues in the original construction. We develop a fast EXIT function computation method that does not rely on Monte-Carlo simulations and can be used in other scenarios as well. Our approach allows us to jointly optimize code performance across the range of rates under puncturing. We then consider protograph E^2RC codes (that have a protograph representation) and propose rules for designing a family of rate-compatible punctured protographs with low thresholds. For both the semi-structured and protograph E^2RC families we obtain codes whose gap to capacity is at most 0.3 dB across the range of rates when the maximum variable node degree is twenty.

In this chapter, we give an overview on the current progress on studying wireless information flow and the proposal of the deterministic channel model, as well as the development of low-density parity-check (LDPC) codes and rate-compatible LDPC codes.

1.1 Introduction on Wireless Information Flow

Information theory by Shannon (1948) has laid the theoretical foundation for modern communications. It says that there is an inherent fundamental limit on the information rate that can be transmitted over any given communication channel, which is called channel capacity. And this theoretical limit should guide the development of any practical transmission schemes. There is no hope to exceed this limit but practical transmission schemes can be designed trying to approach this limit so as to maximize the utility of the communication channels.

Since its advent, Shannon's information theory has been successfully applied in computing the capacities of various point-to-point channels, however, the extension to wireless relay networks has proven to be a really challenging task. In fact, except for a few simple wireless networks, the exact capacity region of most wireless networks is unknown. The difficulty comes from the facts that signal interactions in the wireless networks are very complicated. The inherent features of wireless communications include broadcasting, interfering, fading, etc, and all these features contribute to the difficulty of the problem for finding out the exact capacity regions for wireless networks.

The most widely used physical layer channel model in wireless communications has been the linear channel model with Gaussian noise. Fig. 1.1 shows an example of a Gaussian channel model

$$y = h \cdot x + n$$

where $n \sim \mathcal{N}(0,1)$ is the additive Gaussian noise variable and h is the channel gain.

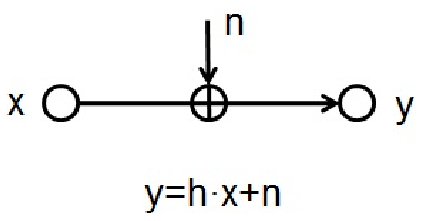


Figure 1.1 Gaussian channel model

1.1.1 The Known Facts

For the Gaussian wireless networks, only for a few network examples, i.e., the point-to-point Gaussian channel, the Gaussian broadcast channel and the Gaussian multiple access channel (MAC), their capacity regions are known exactly(Cover & Thomas (2006)). Next let's recall their capacity characterization.

For a point-to-point Gaussian channel as shown in Fig. 1.1, the capacity is given as

$$C = \frac{1}{2}\log_2(1 + \text{SNR})$$

where SNR= h^2 is the signal to noise ratio of the channel. Here both transmit power and noise power are normalized to be 1 and the signal-to-noise ratio (SNR) is captured in terms of channel gains.

A Gaussian broadcast channel is shown in Fig. 1.2 with

$$y_i = \sqrt{\mathrm{SNR}_i} \cdot x + n_i, i = 1, 2$$

The transmitter wishes to send independent messages at rates R_1 and R_2 to receivers y_1 and y_2, respectively. The capacity region is given as

$$R_1 \leq \frac{1}{2}\log_2(1 + \alpha\mathrm{SNR}_1)$$
$$R_2 \leq \frac{1}{2}\log_2(1 + \frac{(1-\alpha)\mathrm{SNR}_2}{\alpha\mathrm{SNR}_2 + 1})$$

where SNR_1 and SNR_2 are signal to noise ratios of the two link channels respectively and α may be arbitrarily chosen $0 \leq \alpha \leq 1$ to trade off rate R_1 for rate R_2 as the transmitter wishes. Fig. 1.3 shows the capacity region for the case when $\mathrm{SNR}_1 \geq \mathrm{SNR}_2$.

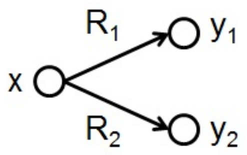


Figure 1.2 Gaussian broadcast channel model

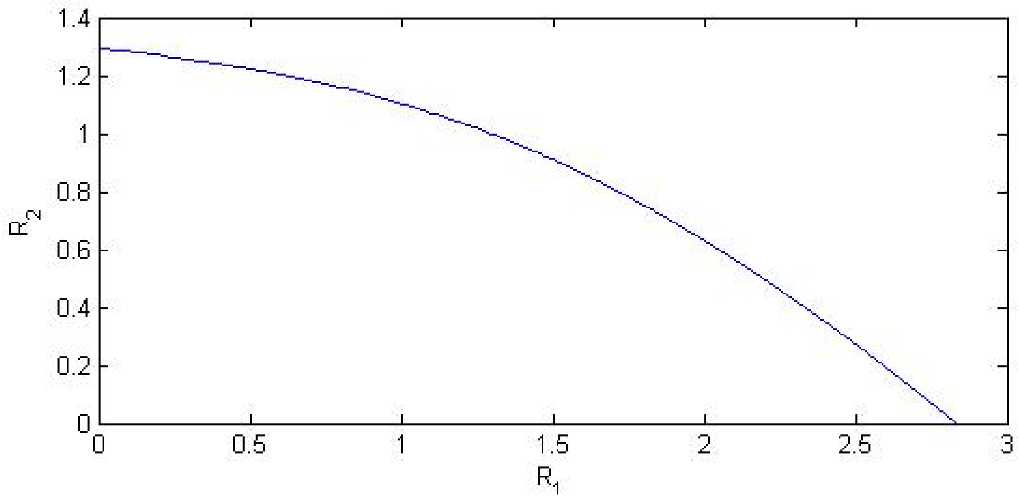


Figure 1.3 Capacity region of Gaussian broadcast channel

A Gaussian MAC is shown in Fig. 1.4 with

$$y = \sqrt{\mathrm{SNR}_1} \cdot x_1 + \sqrt{\mathrm{SNR}_2} \cdot x_2 + n$$

Two transmitters want to send information to a common receiver at rates R_1 and R_2 respectively. The capacity region is given as

$$\begin{aligned} R_1 &\le \frac{1}{2}\log_2(1+\mathrm{SNR}_1) \\ R_2 &\le \frac{1}{2}\log_2(1+\mathrm{SNR}_2) \\ R_1 + R_2 &\le \frac{1}{2}\log_2(1+\mathrm{SNR}_1+\mathrm{SNR}_2) \end{aligned}$$

where SNR_1 and SNR_2 are signal to noise ratios of the two broadcasting channels respectively. Fig. 1.5 shows the capacity region for the case when $\mathrm{SNR}_1 \ge \mathrm{SNR}_2$.

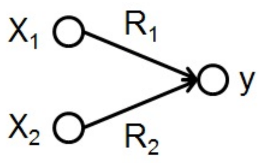


Figure 1.4 Gaussian multiple access channel model

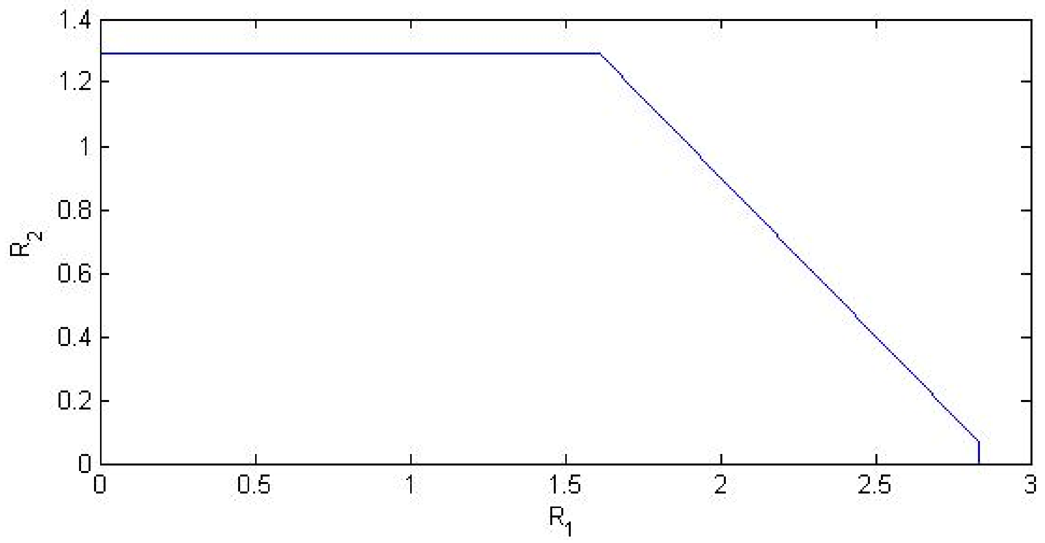


Figure 1.5 Capacity region of Gaussian multiple access channel

1.1.2 The Unknown

The above wireless network examples are the only a few cases when their capacities are known exactly. For all other wireless networks, including the very simple single relay Gaussian network as shown in Fig. 1.6 and two relay Gaussian network as shown in Fig. 1.7, their capacities are not known exactly, let alone more complicated Gaussian wireless networks. In general, the only known upper bound on the capacity of Gaussian relay networks is the information theoretical cut-set upper bound, which says that if the information rates $\{R_{ij}\}$ are achievable, there exists some joint probability distribution $p(x_1, x_2, ..., x_m)$ such that

$$\sum_{i \in S, j \in S^c} R_{ij} \leq I(X_S; Y_{S^c} | X_{S^c})$$

for all $S \subset \{1, 2, ...m\}$, where S/S^c is a cut separating the network nodes 1 through m.

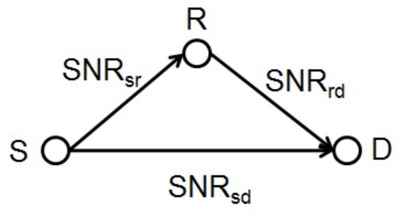


Figure 1.6 Single relay Gaussian network

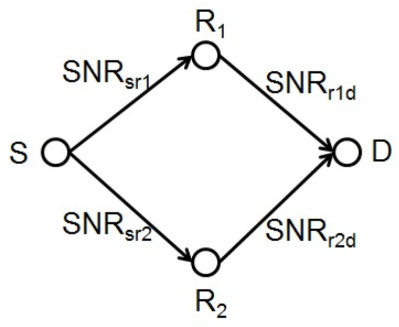


Figure 1.7 Two relays Gaussian network

Several relay strategies for wireless relay networks have been developed, such as amplify-and-forward (AF), decode-and-forward (DF), etc., but none of them achieve this cut-set upper bound. It is even not clear what the gap is between the achievable rates of these relay strategies and the cut-set upper bound.

1.2 A Step Further – the Deterministic Channel Model

To make further progress, Avestimehr, etc. (2007 - 1) have proposed a deterministic channel model (i.e., the ADT model) to approximate Gaussian channel model. It is a far simpler channel model than the Gaussian channel model which make the analysis of wireless information flow simpler. The ADT model not only models the two main features of wireless communications, namely, broadcasting and superposition (or interfering), but also be more analytically tractable. In Chapter 2, we will elaborate on the ADT model and motivate our study of network flow algorithms for deterministic wireless relay networks.

1.3 Introduction on LDPC Codes

LDPC codes introduced in Gallager (1963) have near-capacity performance on a large variety of channels and low decoding complexity, and have been proposed in a number of new applications and standards. An LDPC code is a linear block code with low density non-zero entries in its parity-check matrix.

1.3.1 Representation of LDPC Codes

Like other linear block codes, an LDPC code can be defined by its generator matrix or parity-check matrix and most commonly by its parity-check matrix. Moreover an LDPC code can be equivalently represented by a bipartite graph or the so called Tanner graph in Tanner (1981), where each edge in the graph connects a variable node (representing the bits) on one side with a check node (representing the parity-check equations) on the other side.

1.3.2 Regular and Irregular LDPC Codes

Let M, N be the total number of columns, rows respectively in the parity-check matrix for an LDPC code and $K = N - M$. A regular LDPC code has a parity-check matrix with exactly the same number w_c non-zero entries in each column and exactly the same number w_r non-zero entries in each row, where both w_c and w_r are much smaller than the total number of rows. The code rate of an LDPC code is

$$R = K/N = 1 - w_c/w_r$$

assuming that the parity-check matrix H has full-rank. An irregular LDPC code has a parity-check matrix with varying number of non-zero entries in each column or row. To specify an ensemble of irregular LDPC codes, degree distribution polynomials $\lambda(x)$ and $\rho(x)$ are used.

$$\begin{aligned} \lambda(x) &= \sum_{d=1}^{d_v} \lambda_d x^{d-1} \\ \rho(x) &= \sum_{d=1}^{d_c} \rho_d x^{d-1} \end{aligned}$$

where λ_d denotes the fraction of all edges connected to degree-d variable nodes and d_v denotes the maximum variable node degree, ρ_d denotes the fraction of all edges connected to degree-d check nodes and d_c denotes the maximum check node degree.

1.3.3 Decoding of LDPC Codes

The commonly used decoding algorithms for LDPC codes are some kinds of iterative decoding algorithm, known as belief propagation algorithm, sum-product algorithm or message passing algorithm. There are also some variations of iterative algorithms. These iterative algorithm compute the distributions of variables (likelihood ratios) passing around the code's Tanner graph during the decoding process. Usually a finite number of iterative decoding iterations is enforced on the decoder and the codeword is decoded based on the final distributions of the likelihood ratios.

We describe the iterative decoder for LDPC codes assuming that a binary-input AWGN channel, $y = x + n$, is in use, where $x \in \{+1, -1\}$ corresponds to coded bit $c \in \{0, 1\}$ and $n \sim \mathcal{N}(0, \sigma^2)$ is the additive Gaussian noise variable. The log-likelihood ratio for the coded bit c is computed as

$$L_{ch} = \log\left(\frac{p(x = +1|y)}{p(x = -1|y)}\right) = \frac{2y}{\sigma^2}$$

Let $V(C)$ denote the set of all variable (check) nodes in the Tanner graph representation of the LDPC code. During the decoding process, the message passing follows the following rules:

$$\begin{aligned} L_{v\to c} &= L_{ch,v} + \sum_{c'\in C, c'\neq c} L_{c'\to v} \\ L_{c\to v} &= 2tanh^{-1}(\prod_{v'\in V, v'\neq v} tanh(L_{v'\to c})) \end{aligned}$$

where $L_{v\to c}$ is the message from variable node v to check node c if there is edge connection between them in the Tanner graph and $L_{c\to v}$ is the message from check node c to variable node v if there is a connection between them in the Tanner graph, similarly defined the other variables in the equations above.

Upon termination of the decoding process, the coded bit c is decoded as 0 if its log-likelihood ratio is larger than 0, otherwise is decoded to be 1.

1.3.4 Protograph LDPC Codes

There have been numerous constructions of LDPC codes proposed in the literature ranging from random choice to algebraic constructions (Richardson, etc. (2001) and references herein). Most LDPC codes used in industry need to have some structure that allows parallelizable decoding. Moreover the amount of storage required to store the description of the parity-check matrix needs to be small, i.e., storing completely random permutations is not feasible due to implementation issues. The protograph based LDPC codes introduced in Thorpe (2003) address this issue in part.

Protograph codes have the advantage that the asymptotic threshold of the code can be found by performing density evolution (Richardson, etc. (2001), Richardson & Urbanke (2001), Richardson (2009)) on the protograph. Moreover, if instead of a random permutation we choose a random circulant permutation that can be specified by a circular shift of the identity matrix then the storage requirement can be reduced tremendously and a fast parallelizable decoder can be implemented in hardware.

LDPC codes based on protographs were first introduced in Thorpe (2003). The main idea here is to start with a small mini-graph (called a protograph) and construct the LDPC code by replacing each edge in the protograph by a random permutation of a fixed size (see Figure 1.8 for an example). Protograph based codes can be considered as a subclass of the multi-edge type LDPC codes Richardson (2009). In the protograph representation, each edge represents one edge type, and parallel edges are allowed. Density evolution can be performed on protographs to determine their asymptotic performance.

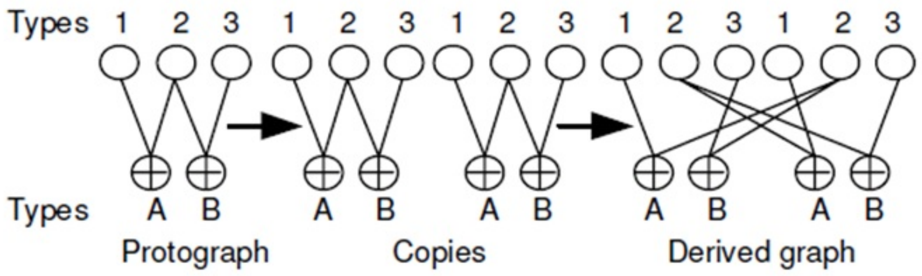


Figure 1.8 Copy-and-permute in generating larger graph from protograph

1.3.5 Multi-Edge Type LDPC Codes

Multi-edge type LDPC codes in Richardson (2009) is a generalization of the concept of irregular LDPC codes that yields improvements in performance, range of applications, adaptability and error floor. The degree distribution polynomial pair, $(\lambda(x), \rho(x))$, for studying irregular LDPC codes apply very well when the LDPC code in consideration is highly random or non-structured. For structured or semi-structured LDPC codes, such as E^2RC codes, RA codes, IRA codes, eIRA codes, etc, multi-edge type LDPC codes proposed in Richardson (2009) would be a better representation, admitting all the above mentioned constructions as special cases. The multi-edge type LDPC codes concept is used throughout our work in Chapter 4, for both semi-structured E^2RC codes and protograph E^2RC codes and proves to be very effective.

1.4 Rate-Compatible LDPC Codes

Another desirable feature in practice especially for wireless channels is rate-compatibility. Rate-compatible puncturing was introduced in Hagenauer (1988) as a technique to obtain a family of codes of varying rates while retaining the same encoder-decoder pair. Rate-compatible punctured codes are a practical low-complexity solution that are useful in hybrid-ARQ protocols and situations where the channel quality varies over time. The mother code (which is systematic) in these systems corresponds to the lowest code rate. Higher code rates can be obtained by only transmitting a subset of the parity bits. The parity bits that are not transmitted are said to be punctured. The parity bits of higher-rate codes are chosen to be a subset of the parity bits of the lower rate codes. Throughout this thesis for the bipartite graph representation of a rate-compatible LDPC code, we follow the convention that a blank circle represents an unpunctured variable node that participates in the transmission and a filled circle represents a punctured variable node that does not participate in transmission. A check node is represented by a blank circle with a plus sign in it.

1.4.1 Construction of Rate-Compatible LDPC Codes

A number of papers have investigated the construction of rate-compatible punctured LDPC codes Kim, etc. (2009), Ha, etc. (2004 - 1), Ha, etc. (2004 - 2), Ha, etc. (2006), Yue, etc. (2007), Yazdani & Banihashemi (2004). The main challenge here is the design of a mother code and the puncturing pattern such that the BER/FER of the codes of all rates is low. In order words, it is desirable that the rate-compatible codes have good code performance at all code rates and the code performance at all code rates can be managed and predicted. There are two main approaches that address the problem of rate compatibility.

- *Optimizing degree distributions for puncturing.*

  In Ha, etc. (2004 - 1), the authors found the optimal degree distributions for puncturing using density evolution analysis. In Ha, etc. (2004 - 2), Ha, etc. (2006), Yue, etc. (2007), the authors proposed algorithms for finding good puncturing patterns for given mother code.

- *Design of a good mother code and the puncturing pattern.*

  Here the attempt is to design LDPC codes with a specific structure that allows good performance across a range of rates. Some recent works include Kim, etc. (2009), Yazdani & Banihashemi (2004). These approaches have been guided in part by the criteria used in design good puncturing patterns as well.

In chapter 2, we will elaborate on the efficiently encodable rate-compatible LDPC codes by Kim, etc. (2009) as our work is inspired by it. We will also introduce the density evolution and EXIT chart techniques that have been useful in the design and analysis of LDPC codes and being used in this thesis.

CHAPTER 2. REVIEW OF LITERATURE

In the first part of this chapter, the background and related work are discussed on the subject of studying wireless information flow using deterministic wireless networks. First, the deterministic channel model is introduced, i.e., the ADT model in Avestimehr, etc. (2007 - 1), Avestimehr, etc. (2007 - 2), Avestimehr, etc. (2009). After that, the characterization of the capacity regions of the deterministic networks are given. The methodologies of adopting the ADT model in studying wireless information flow are also discussed. Second, the current status on the network flow algorithms for studying the capacity of deterministic wireless networks is given. Several related network flow algorithms are recapped.

In the second part of this chapter, we introduce the efficiently-encodable rate-compatible LDPC (E^2RC) codes. It motivates our work on the design and analysis of rate-compatible LDPC codes in Chapter 4. The construction of E^2RC codes is given with brief discussion on the desirable features of the codes. Density evolution including EXIT chart analysis proves to be very useful in the design and analysis of LDPC codes and has been used throughout our work in Chapter 4. They are introduced at the end of this chapter.

2.1 The Deterministic Channel Model (ADT Model)

The complex signal interactions in wireless relay networks challenge the study of wireless information flow for many years. To characterize the capacity and capacity-achieving transmission schemes for wireless relay networks still remain open questions. Towards this end, the deterministic channel model for wireless relay networks proposed by Avestimehr, Diggavi and Tse has been a significant progress. The broadcast and inference are two fundamental features of wireless communications. The deterministic channel model captures the broadcast and inference features of wireless communications in addition to converting the wireless relay networks into deterministic networks. Studying the information flow in the deterministic networks provides a way to find out the approximated capacity and corresponding transmission strategies for original wireless relay networks.

Gaussian channel model has been the most widely used channel model for the point-to-point channels in wireless relay networks. The deterministic channel model quantizes the transmitted signal into different bit levels and at the receiver keeps the signal bit levels above the noise level (which depends on the signal to noise ratio (SNR) of the channel) so as to convert the original Gaussian channel into a deterministic channel without random noise variables. The broadcasting of signal at the transmitter is still preserved in the deterministic channel and the interference of signal at the receiver is modeled by modulo two sum of the bits arrived at the same signal level.

Now let's introduce the deterministic channel model by using the example of a point-to-point AWGN channel from Avestimehr, etc. (2007 - 1). Consider an AWGN channel

$$y = h \cdot x + z$$

where $z \sim \mathcal{N}(0,1)$ is the additive Gaussian noise variable and $h = \sqrt{\text{SNR}}$ is the channel gain. Here both transmit power and noise power are normalized to be 1 and the signal-to-noise ratio is captured in terms of channel gains. Assume x and z are real numbers, then y can be written as

$$y \approx 2^n \sum_{i=1}^{n} x(i)2^{-i} + \sum_{i=1}^{\infty}(x(i+k) + z(i))2^{-i}$$

where $n = \lceil \frac{1}{2} \log \text{SNR} \rceil$. If the transmitted signal x is treated as a sequence of bits at different signal levels, then the deterministic channel model truncates x and passes only its bits above noise level. Fig. 2.1 gives a concrete example. At the transmitter node T_x and receiver node R_x, each small cycle represents a signal level. Assume $n = 4$, so only the first four most significant signal levels or bits from T_x are received at R_x. Accordingly each edge in the model can transmit one-bit information at a time.

Similarly, the deterministic models for Gaussian broadcast channel and Gaussian multiple access channel are shown in Fig 2.2 and 2.3. Note that the broadcasting is modeled by the

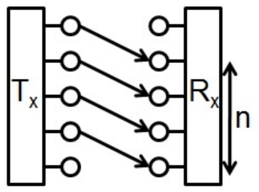


Figure 2.1 The deterministic channel model for a point-to-point Gaussian channel.

fact that all signals send from the same signal level of the transmitter would be the same and the inference or superposition is modeled by the fact that for all signals received at the same signal level of the receiver, only the modulo two sum of them is available to the receiver.

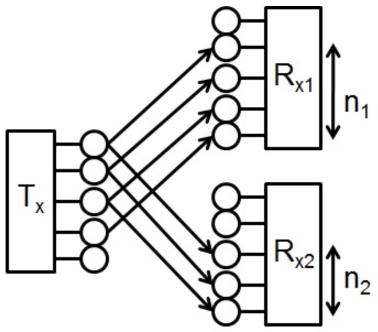


Figure 2.2 The deterministic channel model for Gaussian broadcast channel.

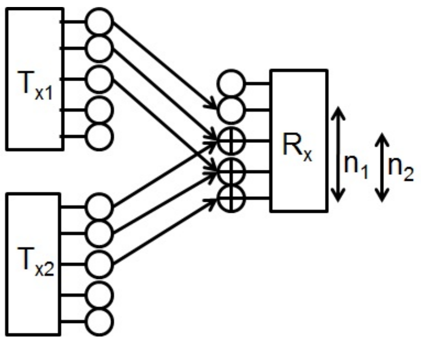


Figure 2.3 The deterministic channel model for Gaussian MAC.

The deterministic channel model described above is called linear finite-field deterministic channel model in Avestimehr, Diggavi and Tse '07, which is referred to as linear deterministic channel model in this report.

2.1.1 How Close Is the ADT Model

In Fig. 2.1, $n = \lceil \frac{1}{2} \log \mathrm{SNR} \rceil$, so it's easy to conclude that the gap between the capacity of the ADT model and the Gaussian channel model is within $\frac{1}{2}$ bit.

From Fig. 2.2, the capacity region of the deterministic broadcast channel is given by

$$R_1 \leq n_1$$

$$R_2 \leq n_2$$

$$R_1 + R_2 \leq n_1$$

where it's assumed $\mathrm{SNR}_1 \geq \mathrm{SNR}_2$ without loss of generality. The gap between the capacity regions of the deterministic broadcast channel and of the Gaussian channel is within 1 bit per user.

From Fig. 2.3, the capacity region of the deterministic multiple access channel is given by (without loss of generality, assume $\mathrm{SNR}_1 \geq \mathrm{SNR}_2$)

$$R_1 \leq n_1$$

$$R_2 \leq n_2$$

$$R_1 + R_2 \leq n_1$$

where it's assumed $\mathrm{SNR}_1 \geq \mathrm{SNR}_2$ without loss of generality. Again the gap between the capacity regions of the deterministic multiple access channel and of the Gaussian MAC is within 1 bit per user.

The negative part about the ADT model is that it fails to approximate the Gaussian MIMO model very well and the gap between the two capacities could be arbitrarily large. Consider a Gaussian MIMO channel with channel matrix

$$H = 2^k \begin{pmatrix} \frac{3}{4} & 1 \\ 1 & 1 \end{pmatrix}$$

For the Gaussian model, the singular value of H is of the order of 2^k, so the capacity of the Gaussian MIMO channel is of the order of $2 \times \frac{1}{2} \log(1 + |2^k|^2) \approx 2k$. For the deterministic model, $N_{11} = N_{12} = N_{21} = N_{22} = k$, so the capacity of the deterministic channel is k. For this example, the gap between the capacities is unbounded when k grows.

Even through the deterministic channel model doesn't approximate the Gaussian channel model very well in all cases, it's still useful in providing insights in our study of wireless information flow in many scenarios as will become clearer in the later discussion.

2.1.2 Characterization of the Capacity Region of the Deterministic Networks

For general Gaussian relay networks, the cut-set upper bound gives the upper limit on code rate R that allows reliable transmission of information. Let $x_i, y_i, 1 \leq i \leq n$ be the transmitted signals and received signal at node i in the network. Let Ω be a cut separating the nodes in the network into two parts Ω and Ω^c. The cut-set upper bound says the information transmission rate from $S \subseteq \Omega$ to $D \subseteq \Omega^c$ is bounded by

$$C_{SD} \leq \max_{p(x_1,\ldots,x_n)} \min_{\Omega} I(X_\Omega; Y_{\Omega^c}|X_{\Omega^c}) \tag{2.1}$$

In the deterministic networks, the unicast capacity and multicast capacity actually achieve this upper bound.

Since in the deterministic network, the relationship between inputs and outputs are deterministic, the cut-set upper bound in equation 2.1 is equal to

$$C_{SD} \leq \max_{p(x_1,\ldots,x_n)} \min_{\Omega} H(Y_{\Omega^c}|X_{\Omega^c}) \tag{2.2}$$

2.1.2.1 Unicast capacity

Theorem 1 *Given a linear finite-field relay network (with broadcast and multiple access), the unicast capacity C_{SD} (S - source, D - destination) of such a relay network is given by,*

$$C_{SD} = \min_{\Omega} rank(\mathcal{G}_{\Omega,\Omega^c})$$

where the minimum is taken over all cuts separating S from D in $\mathcal{G}$.

Proof

Refer to Avestimehr, etc. (2007 - 2) for proof details. Here only the main ideas of the proof are reviewed.

The proof is first done on networks that have a layered structure, i.e. all paths from the source to the destination have equal lengths. With this special structure we get a major simplification: a sequence of messages can each be encoded into a block of symbols and the blocks do not interact with each other as they pass through the relay nodes in the network.

Proof outline for layered networks is given below.

The encoding for layered linear deterministic relay networks goes as follows. Source S has a sequence of messages

$$w_k \in \{1, 2, ..., 2^{TR}\}, k = 1, 2, ...$$

Each message is encoded by the source S into a signal over T transmission times (symbols), yielding an overall transmission rate of R. Each relay j operates over blocks of time T symbols using a mapping

$$f_j : \mathcal{Y}_j^T \to \mathcal{X}_j^T$$

on its received symbols from the previous block of T symbols to transmit signals in the next block. The mapping is

$$x_j = F_j y_j$$

where F_j is chosen uniformly randomly over all matrix in $\mathcal{F}_2^{qT \times qT}$. Given all encoding functions F_j, the decoder D$\in \mathcal{D}$ attempts to decode each message w_k sent from the source.

The probability of error at decoder D is upper bounded by

$$P_e \leq 2^{RT}(P(w \to w') = 2^{RT} P(y_d(w) = y_d(w')) \tag{2.3}$$

where

$$P(w \to w') = P(y_d(w) = y_d(w'))$$

is the probability that the decoder D mistakes the message w for another message w', which can be written as

$$P(w \to w') = \Sigma_{\Omega \in \Lambda_D} P(\text{Nodes in } \Omega \text{ can distinguish } w, w' \text{ and nodes in } \Omega^c \text{ cannot}) \tag{2.4}$$

G_{Ω,Ω^c} is defined as the transfer matrix associated with the nodes in Ω to the nodes in Ω^c.

For a layered deterministic network, this transfer matrix breaks up into block diagonal elements corresponding to each layer of the network. There are L (total number of layers of nodes in the network) disjoint sub-network corresponding to each layer with $\beta_l(\Omega)$ nodes at distance $l-1$ from S that are in Ω on one side and $\gamma_l(\Omega)$ nodes at distance l from S that are in Ω^c on the other, $1 \leq l \leq L$. For all the receivers in γ_l, the vector of received signal is denoted as $z_l(w)$, we have

$$z_l(w) - z_l(w') = I_T \otimes G_l[u_l(w) - u_l(w')], l = 1, 2, ...L$$

where the transmitted signals from $\beta_1, ...\beta_L$ are clubbed together and denoted by $u_l(w), l = 1, 2, ...L$. The probability that $z_l(w) = z_l(w')$ is the probability that $u_l(w) - u_l(w')$ lies in the null space of $I_T \otimes G_l, l = 1, 2, ...L$.

$$P\{u_l(w) - u_l(w') \in \mathcal{N}(I_T \otimes G_l)\} = p^{-\text{rank}(I_T \otimes G_l)} = p^{-T\text{rank}(G_l)}$$

$$P\{u_l(w) - u_l(w') \in \mathcal{N}(I_T \otimes G_l), l = 1, 2, ...L\} = \prod_{l=1}^{L} p^{-\text{rank}(I_T \otimes G_l)} = p^{-T\sum_{l=1}^{L}\text{rank}(G_l)}$$

Finally,

$$P(w \to w') \leq \sum_{\Omega \in \Lambda_D} p^{-T\sum_{l=1}^{L}\text{rank}(G_l(\Omega))} \leq 2^{|\mathcal{V}|} p^{-T\min_{\Omega\in\Lambda}\text{rank}(G_{\Omega,\Omega^c})}$$

Put it into equation 2.3, we can drive the error probability to zero if

$$R < \min_{\Omega \in \Lambda_D} \text{rank}(G_{\Omega,\Omega^c})$$

The above proof is easily extended to the case of multicast.

What left is to extend the current proof to arbitrary deterministic networks besides layered deterministic networks, which is skipped in this thesis (refer to Avestimehr, etc. (2007 - 2) for more details). But here we will introduce the concept of time expanded networks which can expand a non-layered network into a layered network over time.

Time expansion technique

Time expansion of a network is a technique to expand a given network over time. The concept of time expanded network has been used in Ahlswede, etc. (2000) and Avestimehr, etc. (2007 - 2). In the former, it is used to handle cycles. In the latter, it is used to handle interaction between messages transmitted at different times, an issue that only arises when there is superposition of signals at nodes.

Next let's look at an example of the technique of time expansion of a given network. Fig. 2.4 shows an example of general deterministic networks that are not layered. Fig. 2.5 shows the corresponding time expanded deterministic network. Obviously the time expanded deterministic network is a layered network. In the time expanded network, the index $i, 1 \leq i \leq k$ corresponds to the time index or time parameter. There are $k+1$ layers and each node $v \in \mathcal{V}$ appears at all time instants $i, 1 \leq i \leq k$. The auxiliary nodes $T[i]$'s and $R[i]$'s are just virtual transmitters and receivers that are put to buffer and synchronize the network. Since all the communication links connected to $T[i]$'s and $R[i]$'s are modeled as wireline links without capacity limit, they would not impose any constraint on the network. The following is a Lemma from Avestimehr, etc. (2007 - 2). It's cited here without proof.

Lemma 2 *Assume $\mathcal{G}$ is a general deterministic network and $\mathcal{G}^K_{unf}$ is a network obtained by unfolding $\mathcal{G}$ over K time steps. Then the following communication rates is achievable in $\mathcal{G}$:*

$$R < \frac{1}{K} R_{unf}$$

with

$$R_{unf} = \max_{\prod_{i \in \mathcal{V}} p(x_i)} \min_{\Omega_{unf} \in \Lambda_D} H(Y_{\Omega^c_{unf}} | X_{\Omega^c_{unf}})$$

is the achievable rate in the time expanded network where the minimum is taken over all cuts Ω_{unf} in $\mathcal{G}^K_{unf}$.

2.1.2.2 Multicast capacity

The characterization of the multicast capacity of deterministic networks is given. The proof is similar to the above unicast case. Refer to Avestimehr, etc. (2007 - 2) for a complete proof.

Theorem 3 *Given a linear finite-field relay network (with broadcast and multiple access), the multicast capacity C of such a relay network is given by,*

$$C = \min_{D \in \mathcal{D}} \min_{\Omega} rank(\mathcal{G}_{\Omega,\Omega^c})$$

where the first minimum is taken over all cuts in $\mathcal{G}$.

From above discussion, it's easy to conclude that the capacity characterization of the deterministic networks for unicast and multicast sessions has a simple max-flow min-cut form except that here the cut value is defined to be the rank of the binary adjacency matrix for each cut. This difference makes the algorithms for finding the max flow or min cut for normal graphs and for deterministic networks different which will be discussed in Chapter 3.

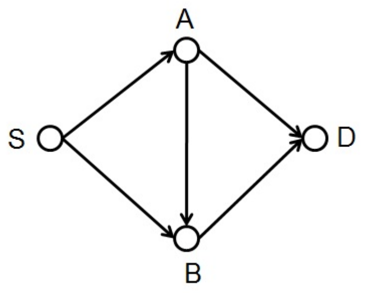


Figure 2.4 An example of general deterministic networks.

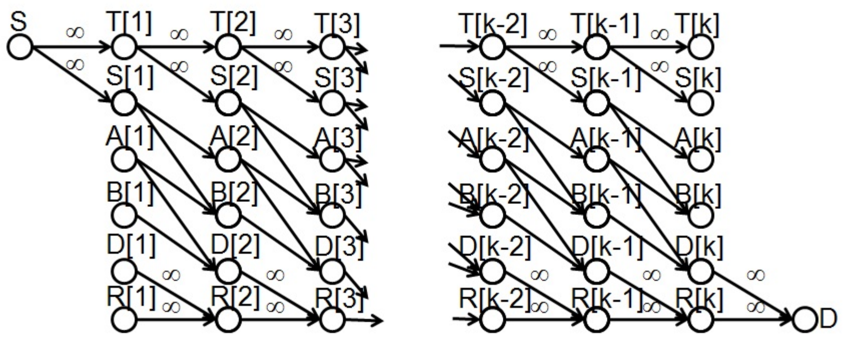


Figure 2.5 The time expanded deterministic network.

2.1.3 Methodologies in Adopting the ADT Model

As mentioned before, due to the complexity of signal interactions, except for the simplest networks such as the one-to-many Gaussian broadcast channel and the many-to-one Gaussian multiple access channel, the capacity of most Gaussian networks is unknown. The deterministic channel model provides a way to make progress in this problem in two steps.

First apply the deterministic channel model and study the information flow in the deterministic networks. The deterministic channel model models the broadcasting and superposition of wireless communications. The motivation for such channel model is that wireless networks often operate in interference-limited regime where the noise power is small compared to signal powers. The deterministic model focuses more on the signal interactions rather than the noise. For the deterministic networks, we have a complete characterization of the capacity for the unicast and multicast cases which is a generalization of the max-flow min-cut theorem for wireline networks.

Second apply the insights gained from the deterministic approach to find near-optimal communication schemes for original Gaussian relay networks. Avestimehr, Diggavi and Tse proved that in the noisy Gaussian network, an approximate max-flow min-cut result holds where the approximation is within an additive constant universal over channel parameters but only depends on the number of nodes in the networks. This is a strong result since none of the existing relay strategies in the literature yield such a universal approximation for arbitrary networks. Avestimehr, Diggavi and Tse using the insights gained from the deterministic networks proposed a quantize-and-forward scheme yielding such a universal approximation.

2.2 Network Flow Algorithms for Deterministic Wireless Networks

There is complete characterization of the capacity region for the deterministic networks for unicast and multicast sessions. However the characterization itself doesn't suggest any efficient algorithms for finding the capacity or corresponding capacity-achieving transmission schemes. An intuitive idea would be to enumerate all source-destination cuts and compare their cut values but this leads to an algorithm with complexity growing exponentially with the size of the networks. So efficient algorithms for identifying the capacity and capacity achieving transmission schemes is desirable. There are the following results on the network flow algorithms for deterministic networks. The first algorithm by Amaudruz & Fragouli (2009) and our algorithm both fall into the category of path augmentation algorithms and the latter can be seen as an improved version of the former, we will introduce their algorithm in details while for other algorithms on the same problem, the readers are referred to the reference paper for more details.

2.2.1 Path Augmentation Algorithms for Wireless Deterministic Networks

The first efficient algorithm addressing the problem of finding unicast capacity for deterministic networks is by Amaudruz & Fragouli (2009). They proposed a polynomial-time algorithm for finding the unicast capacity of any given linear deterministic wireless network by trying to identify the maximum number of linearly independent paths in the network using the idea of path augmentation.

Amaudruz and Fragouli's algorithm is basically a path augmentation algorithm which operates in iterations and in each iteration tries to find an additional path in addition to already found paths in previous iterations. The algorithm is a recursive algorithm and the basic operation is the exploration to a node in the network which is described below.

To explore a node A in the deterministic network, they sequentially examine all signal levels $x_i, x_i \in A$ and take the following actions:

1. If x_i is already used by a path, do nothing.
2. If x_i is not used, then examine each y_j with $(x_i, y_j) \in E$ as follows:
   (a) If y_j is used by some path, find a smallest set L_{x_i} such that replacing each element in this set with x_i, there is still the same number of S-D paths in the network. Then replace each element in the set with x_i and try to find a path starting from the replaced element.

(b) If y_j is not used, then determine the rank of the binary adjacency matrix for the current layer where x_i is. If the matrix is not full rank, do nothing, otherwise, extend the current path to y_j along edge (x_i, y_j). If $\mathcal{A}(y_j) = D$, then the algorithm succeeds and returns one more S-D path. If it turns out that there is no path from y_j to D, then a ϕ function is called to rewire the connections in the current layer where x_i is to replace y_j with an equivalent y'_j such that $\mathcal{A}(y_j) = \mathcal{A}(y'_j)$ and (x'_i, y'_j) is used by some path.

The algorithm has a computational complexity bounded by $O(M \cdot |E| \cdot C^5)$ where M is the maximum number of nodes in one layer, E is the total number of edges in the network and C is the unicast capacity of the underlying network. As will be seen later in Chapter 3, though our algorithm and the algorithm by Amaudruz and Fragouli fall into the same category, our algorithm enjoys much less computational complexity than the latter.

2.2.2 Other Combinatorial Algorithms for Wireless Deterministic Networks

Other efficient algorithms for addressing the same problem of finding unicast capacity of deterministic networks include that by Yazdi & Savari (2009) and that by Goemans, etc. (2009).

Yazdi & Savari (2009) developed a two-dimensional Rado-Hall transversal theorem for block matrices and used the submodularity of the capacity of a cut to formulate the problem as a linear program over the intersection of two polymatroids so as to solve the problem in polynomial time. Their algorithm has a computational complexity bounded by $O(L^8 \cdot M^{12} \cdot h_0^3 + L \cdot M^6 \cdot C \cdot h_0^4)$, where L is the total number of layers in the layered deterministic network and h_0 is the maximum number of signal levels from all nodes in one layer. Refer to Yazdi & Savari (2009) for more details.

Along the same line, Goemans, etc. (2009) has proposed another efficient algorithm for tackling the same problem. First they formulate the deterministic networks as specific instances of linking systems, a combinatorial structure with a tight connection to matroids so that properties and algorithms available for matroids and linking systems can be extended to solve this problem. They prove the max-flow min-cut theorem and submodularity of cuts from the linking system aspect. Finally, efficient algorithms for matroids intersection or matroids partition is used to find a maximum flow or minimum cut in the network. The computational complexity for this algorithm is bounded by $O(L^{1.5} \cdot M^{3.5} \log ML)$ or $L \cdot M^3 \cdot \log M$, which is a strong polynomial time algorithm and would be much faster for large capacity deterministic networks.

2.3 Efficiently Encodable Rate-Compatible LDPC (E^2RC) Codes

A significant amount of research work has dealt with the construction and analysis of punctured LDPC codes Kim, etc. (2009), Ha, etc. (2004 - 1), Ha, etc. (2004 - 2), Ha, etc. (2006), Yazdani & Banihashemi (2004), Yue, etc. (2007). In this section we briefly explain the construction introduced by Kim, etc. (2009) as our work is inspired by it. Let the parity-check matrix of a systematic LDPC code be denoted

$$H = [H_1|H_2]$$

where H_1 denotes the sub-matrix corresponding to the information bits and H_2 denotes the sub-matrix corresponding to the parity bits. We say that a parity node in H_2 is k-step recoverable (or $k - SR$) if it can be recovered in exactly k iterations of iterative decoding assuming that all the parity bits are punctured and all the information bits are unpunctured (Figure 4.2 shows an example). Intuitively, a large number of low-SR nodes tend to reduce the required number of decoding iterations in the high SNR regime and result in good puncturing performance.

Efficiently-Encodable Rate-Compatible (E^2RC) codes have a linear-time encoder and have good performance under puncturing for a wide variety of rates. In Kim, etc. (2009) the submatrix H_2 consists of exclusively degree-2 and degree-1 nodes. Moreover half the nodes in H_2 are $1 - SR$, one-fourth are $2 - SR$ and so on[1]. The special structure of H_2 for E^2RC codes results in good puncturing performance with a simple puncturing pattern. The puncturing pattern is such that $1 - SR$ nodes should be punctured first and then the $2 - SR$ nodes and so on depending upon the rate requirement.

[1]This is strictly true if the number of parity nodes is a power of two and approximately true otherwise.

The design goal of E^2RC codes is to obtain rate-compatible punctured codes that exhibit good code performance across a wide range of code rates. E^2RC codes are a class of systematic irregular LDPC codes. Assuming a full-rank parity-check matrix, there are K columns corresponding to information bits and M columns corresponding to parity bits, with $K + M = N$. In the sequel the submatrix of the parity-check matrix corresponding to the K information bits (H_1) is referred to as systematic part and the submatrix corresponding to the parity bits is referred to as non-systematic part (H_2).

Define depth d as the number of different types of k-SR matrices that have degree-2 columns in a parity-check matrix. Let the function $\gamma(k)$ be the number of columns in the k-SR matrix in a parity-check matrix. The construction of E^2RC codes are briefly described below. The readers are referred to Kim, etc. (2009) for details.

- *Finding optimal degree distribution*

  This step is to find the optimal degree distribution for the desired mother code rate with constraint that number of degree-2 check nodes is smaller than M.

- *Parameter setting*

  For the given design parameter M, find out d and $\gamma(k)$ to determine the exact structure of the non-systematic part.

- *Generating k-SR matrix*

  The j-th column of the k-SR matrix has the following sequence:

$$\begin{aligned} h_{k,j} &= D^{j+S_{k-1}}(1 + D^{\gamma(k)}), 1 \leq k \leq d, 0 \leq j \leq \gamma(k) - 1 \\ &= D^{M-1}, k = d + 1, 0 \leq j \leq \gamma(k) - 1 \end{aligned}$$

  where S_k represents the sum of the number of columns in the submatrix formed by the placing the 1-SR, 2-SR, and k-SR matrices next to each other and D^i represents the position of nonzero element in a column, i.e., i-th element of the column is nonzero, where $0 \leq i \leq M - 1$.

- *Constructing matrix T as follows*

  T = [1-SR matrix|2-SR matrix|...|d-SR matrix]

- *Forming matrix H_2*

  By adding a degree-1 node to T, that is,

$$H_2 = [T|(d+1)\text{-SR matrix}]$$

- *Edge construction*

  Construct H_1 by matching the degree-distribution obtained in step 1 as close as possible.

- *Construct matrix H*

$$H = [H_1|H_2]$$

The E^2RC codes has the following desired features:

- Degree-2 variable nodes are assigned to non-systematic bits only,

- No-cycles involve only degree-2 variable nodes,

Moreover it is possible to avoid cycles of length four in E^2RC codes by making use of the algorithms in Hu, etc. (2001), Tian, etc. (2004), Ramamoorthy & Wesel (2004) and Weng, etc. (2004).

The E^2RC codes designed above can be encoded in linear time which is another desirable feature in practice.

In the original construction of E^2RC codes Kim, etc. (2009), an optimal degree distribution pair (λ, ρ) for unstructured irregular LDPC codes is used for constructing $H = [H_1|H_2]$ of mother code. H_2 is constructed according to designed structure and H_1 is constructed by attempting to match the chosen pair (λ, ρ), i.e., the structure of H_2 is not taken into account in designing the optimal degree distribution. In Chapter 4, we design new classes of E^2RC-like codes, where the structure of H_2 is taken into account in the design process. We will show that families of E^2RC-like codes are designed, exhibiting uniformly good code performance across a wide range of code rates.

2.4 Density Evolution

Density evolution developed by Richardson, etc. (2001), Richardson & Urbanke (2001), is a numerical technique for analyzing the asymptotic performance of modern codes employing iterative decoders. The asymptotic performance of a code is indicated by a threshold value of the channel parameter. Any channel with a worse channel parameter cannot expect to have zero-error approaching transmission performance with the given code. While any channel with a better channel parameter can expect to have zero-error approaching transmission performance with the given code when the code length goes to infinity and the number of iterative decoding iterations goes to infinity.

Density evolution refers to the evolution of the probability density function of the various quantities passed around in the code's Tanner graph during its decoding process. Using density evolution, Richardson, etc. (2001) designed an irregular LDPC code whose asymptotic performance is within 0.06dB from capacity and Chung (2000) designed an irregular LDPC code whose asymptotic performance is within 0.0045 dB from capacity on a binari-input AWGN channel.

In the Appendix B of Richardson & Urbanke (2008), there is a detailed description of efficient implementation of density evolution. In Appendix 5, we give our implementation of the density evolution for computing the decoding threshold of any given protographs.

2.5 EXtrinsic Information Transfer Chart (EXIT Chart) Overview

The convergence behavior of iterative decoders of LDPC codes can be evaluated by tracking the probability distributions of extrinsic log-likelihood ratios (Gallager (1963)). Density evolution technique is developed for such purpose. EXIT chart analysis can be viewed as a sub-category of density evolution techniques, developed by Brink (2001).

EXIT chart describes the flow of extrinsic information through the soft in/soft out constituent decoders in terms of mutual information. The exchange of extrinsic information is highly visualized as a decoding trajectory in the chart. The extrinsic log-likelihood value is the a posteriori log-likelihood value minus the a priori log-likelihood value. An EXIT chart describes the relationship between mutual information I_A and I_E, where I_A is the average mutual information between the bits on the decoder graph edges and the a priori log-likelihood values and I_E is the average mutual information between the bits on the graph edges and the extrinsic log-likelihood values.

For an irregular binary LDPC code transmitted over binary AWGN channel, the EXIT function of a degree d_v variable node is:

$$I_{E,V}(I_A, d_v, \sigma_{ch}^2, R) = J(\sqrt{(d_v-1)[J^{-1}(I_A)]^2 + \sigma_{ch}^2})$$

the EXIT function of a degree d_c check node is:

$$I_{E,c}(I_A, d_c) = 1 - J(\sqrt{(d_c-1)}J^{-1}(1-I_A))$$

where $\sigma_{ch}^2 = 8R \cdot \mathrm{SNR}$.

CHAPTER 3. IMPROVED COMBINATORIAL ALGORITHMS FOR WIRELESS INFORMATION FLOW

The work of Avestimehr, etc. (2007 - 1) has recently proposed a deterministic model for wireless networks and characterized the unicast capacity C of such networks as the minimum rank of the adjacency matrices describing all possible source-destination cuts. Amaudruz and Fragouli first proposed a polynomial-time algorithm for finding the unicast capacity of a linear deterministic wireless network in their 2009 paper. In this work, we improve upon Amaudruz and Fragouli's work and further reduce the computational complexity of the algorithm by fully exploring the useful combinatorial features intrinsic in the problem. Our improvement applies generally with any size of finite fields associated with the channel model. Comparing with other algorithms on solving the same problem, our improved algorithm is very competitive in terms of complexity.

3.1 Introduction

The deterministic channel model for wireless networks proposed by Avestimehr, etc. (2007 - 1), Avestimehr, etc. (2007 - 2) (referred to as ADT model thereafter) has been a useful tool for understanding the fundamental limitations of information transfer in wireless networks. The ADT model captures two main features, the broadcasting and interference, that are present in wireless networks. It converts the wireless networks into deterministic networks, by making appropriate assumptions, that in turn lead to approximate capacity results. The model is called the linear finite-field deterministic channel model in Avestimehr, etc. (2007 - 1), Avestimehr, etc. (2007 - 2). We refer to it as the ADT model and denote the finite field of size p associated with the ADT model as F_p in this chapter.

In Avestimehr, etc. (2007 - 1), Avestimehr, etc. (2007 - 2), the unicast (i.e., with one source S and one destination D) capacity C of any linear deterministic wireless relay network was characterized as the minimum rank of the adjacency matrices describing all its S-D cuts. An exhaustive search for finding the minimum rank of the adjacency matrix for all S-D cuts results in an algorithm with complexity exponential in the size of the network.

Amaudruz & Fragouli (2009) were the first to propose a polynomial-time algorithm for finding the unicast capacity of a linear deterministic wireless relay network (see also Ebrahimi & Fragouli (2009)). In this work, we improve upon Amaudruz and Fragouli's work and further reduce the computational complexity of the algorithm by fully exploring the useful combinatorial features intrinsic in the problem. Our improvement applies generally with any size of finite fields F_p associated with the ADT model. Comparing with other algorithms on solving the same problem by Yazdi & Savari (2009) and by Goemans, etc. (2009), our improved algorithm is very competitive in terms of complexity.

3.2 Notations and Definitions

In Avestimehr, etc. (2007 - 2), it is shown that an arbitrary deterministic wireless network can be expanded over time to generate an asymptotically equivalent (in terms of transmission rate) layered network. Therefore, we focus on layered deterministic networks.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a layered deterministic wireless relay network where $\mathcal{V}$ represents the set of nodes in the original wireless relay network, each node in $\mathcal{V}$ has several different levels of inputs and outputs and $\mathcal{E}$ is the set of directed edges going from one input of some node to one output of some other node. For example, Fig. 3.1(a) gives a graph representation of a layered deterministic wireless relay network where each node is labeled with a capital letter, all inputs (outputs) from nodes are labeled as $\{x_i\}$ ($\{y_j\}$), $1 \leq i, j \leq 8$. In the layered network $\mathcal{G}$, all paths from the source node S to the destination node D have equal lengths (Avestimehr, etc. (2007 - 2)). The set of nodes $\mathcal{V}$ are divided into different layers according to their distances to S. The first layer consists of S and the last layer consists of D. Let $\mathcal{A}(x_i)$ (or $\mathcal{A}(y_j)$) denote the node where an input x_i (or an output y_j) belongs to. Let $\mathcal{L}(A)$ (or $\mathcal{L}(x_i)$, $\mathcal{L}(y_j)$) denote the layer number where node A (or x_i, y_j) belongs to. Denote M as the maximum number of nodes in each layer, L the total number of layers and d the maximum number of outgoing edges from any input in any node in the network $\mathcal{G}$ in this chapter.

A cut Ω in $\mathcal{G}$ is a partition of the nodes $\mathcal{V}$ into two disjoint sets Ω and Ω^c such that $\mathrm{S} \in \Omega$ and $\mathrm{D} \in \Omega^c$. A cut is called a layer cut if all edges across the cut are emanating from nodes from the same layer, otherwise it is called a cross-layer cut. An edge $(x_i, y_j) \in \mathcal{E}$ belongs to layer cut l if $\mathcal{L}(x_i) = l$.

The adjacency matrix $T(\mathbf{x}, \mathbf{y})$ for the sets of inputs $\mathbf{x} = \{x_1, x_2, ...x_m\}$ and of outputs $\mathbf{y} = \{y_1, y_2, ...y_n\}$ in $\mathcal{G}$ is a matrix of size $m \times n$ with binary $\{0, 1\}$ entries. The rows correspond to $\{x_i \in \mathbf{x}\}$ and columns corresponding to $\{y_i \in \mathbf{y}\}$ and $T(i, j) = 1$ if $(x_i, y_j) \in \mathcal{E}$. The adjacency matrix $T(E)$ for a set of edges, E, is the adjacency matrix for the sets of their inputs and their outputs.

A set of edges, E, are said to be linearly independent (LI) if $\mathrm{rank}(T(E)) = |E|$ (where the rank is computed over GF(2)), otherwise they are said to be linearly dependent (LD). In $\mathcal{G}$, each S-D path is of length $L - 1$ and crosses each layer cut exactly once. A set of S-D paths are said to be LI if the subsets of their edges crossing each layer cut are LI, otherwise they are said to be LD. In this work, we will consider a slightly more general adjacency matrix, where the non-zero entries can be from a finite field F_p, and the rank is also computed over F_p. Of course, all our results will also apply to the binary field case.

Let $\mathcal{E}_\Omega$ be the set of edges crossing the cut Ω in $\mathcal{G}$. The cut value of Ω is defined as $\mathrm{rank}(T(\mathcal{E}_\Omega))$, which based on the definition equals the maximum number of LI edges in $\mathcal{E}_\Omega$. Note that the cut value defined above is different than that for regular graphs (which is just the number of edges crossing the cut). It is proved Avestimehr, etc. (2007 - 1), Avestimehr, etc. (2007 - 2) that the unicast capacity of a linear deterministic wireless relay network is equal to the minimum cut value among all S-D cuts.

3.3 Improved Unicast Algorithm

In this section we outline certain improvements that can be made to the algorithm of Amaudruz & Fragouli (2009). In particular, we elaborate on several useful combinatorial aspects that allow us to reduce the overall time complexity. Moreover, these improvements also fix certain issues with the original algorithm by Amaudruz & Fragouli (2009). As mentioned previously, our proposed improvements apply over arbitrary finite fields.

3.3.1 Improving the Original Algorithm

The main idea in Amaudruz & Fragouli (2009) is to find path $\mathcal{P}_{k+1}$ in iteration $k+1$ while maintaining linear independence among all S-D paths in $\mathcal{P}$. In this process, previous paths may be rewired. However, there are cases when the original algorithm may fail to find the exact unicast capacity. We illustrate this using the following examples. We point out that these issues seem to have been resolved in Ebrahimi & Fragouli (2009). However, our proposed algorithm has several differences from Ebrahimi & Fragouli (2009) as discussed in Section 3.4.

3.3.1.1 Improved backward rewiring

We use the example in Fig. 3.1 to show that there are cases where the ϕ-function above is insufficient, causing failures of the original algorithm. Then we illustrate how it can be fixed by introducing an improved backward rewiring mechanism.

In Fig. 3.1(a), three LI S-D paths with color red, green and blue are found in the first three iterations of the algorithm. Let's see how the algorithm goes in iteration four. Let's say the algorithm has extended $\mathcal{P}_4$ along the purple path to y_{20}. The call $E_A(\mathcal{G}, \mathcal{P}, \mathcal{M}, N)$ fails since the only input x_{24} of N is used by paths in $\mathcal{P}$. So ϕ-function is called on y_{19} and then node I is explored in $E_A(\mathcal{G}, \mathcal{P}, \mathcal{M}, I)$, but since there is only one path from all inputs of I to D, $E_A(\mathcal{G}, \mathcal{P}, \mathcal{M}, I)$ fails, and finally the algorithm returns false and reports unicast capacity of 3. However, the unicast capacity of the network is 4 and a capacity-achieving transmission scheme is given by the four S-D paths in Fig. 3.1(b) in different colors.

We propose the following improved backward rewiring mechanism to fix the problem above and to replace the original ϕ-function. Let A denote a node in the network (not to be confused with A in the figure).

First, the backward rewiring is allowed on every node A whenever it is explored in finding $\mathcal{P}_{k+1}$.

Second, the backward rewiring on node A includes the following operations. Let $\mathcal{L}(A) = l+1$. For any output y of A with $y \in U_y^l$ and y is used by a path in $\mathcal{P}$ at the beginning of the current iteration (if such y exists),

- Find one $x \in U_x^l$ such that $T(U_x^l - x, U_y^l - y)$ has full rank,
- Then rematch $(U_x^l - x, U_y^l - y)$ to generate a new set of k LI used path edges in layer cut l and
- Finally try to complete the partial path from $\mathcal{A}(x)$.

Lemma 6 guarantees that for a given $y \in U_y^l$ there is always one such x and also a set of edges[1]

$$P_{y\to x} = \{(x_1, y_1 = y), (x_1, y_2), (x_2, y_2), ...(x_{m'-1}, y_{m'}), (x_{m'} = x, y_{m'})\} = \{e_1, e_2, ..., e_{2m'-1}\}$$

with $(x_i, y_i), 1 \leq i \leq m'$ being edges used by $\mathcal{P}$, which can be found with complexity $O(k^3)$ and $O(k^2)$ respectively. Along the alternating path $P_{y\to x}$, the rematching of the used path edges in layer cut l can be done easily as follows:

$$U^l = U^l - e_1 + e_2 - e_3 + ... - e_{2m'-1}$$

Consider applying our improved backward rewiring in the example in Fig. 3.1. It happens on the outputs of nodes N and I. Its application to N is straightforward. Let's look at its application at the output y_{14} of node I. First it finds $x_6 \in U_x^2$ with $T(U_x^2 - x_6, U_y^2 - y_{14})$ having full rank and the alternating path

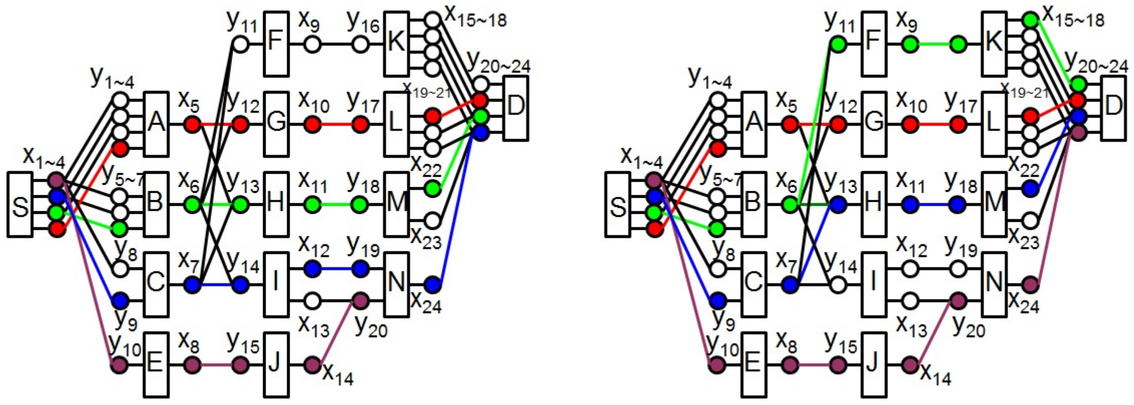
$$P_{y_{14}\to x_6} = \{(x_7, y_{14}), (x_7, y_{13}), (x_6, y_{13})\}$$

[1]We use the notation $P_{y\to x}$ since this set of edges can be interpreted as an alternating path, as we show in Section 3.3.2

. The rematching is done by

$$U^2 = U^2 - (x_7, y_{14}) + (x_7, y_{13}) - (x_6, y_{13})$$

. Then node $B = \mathcal{A}(x_6)$ is explored. Finally the improved algorithm returns four LI S-D paths in Fig. 3.1(b) as expected.



(a) (b)

Figure 3.1 Illustrating example for improved backward rewiring

3.3.1.2 Improved same-layer rewiring

We use the example in Fig. 3.2 to show that the same-layer rewiring in original algorithm is insufficient. Suppose the red S-D path is found in the first iteration. In iteration two, suppose that the algorithm first extends $\mathcal{P}_2$ along the green path to x_4. The same-layer rewiring from x_4 will mark x_3. Since $T(x_3 + x_4, y_5 + y_6)$ is not full rank, the algorithm fails to complete $\mathcal{P}_2$ along the green path. It continues to extend $\mathcal{P}_2$ along the blue path to x_5. Since x_3 is marked, the same-layer rewiring from x_5 won't be applied on x_3 and the call $E_A(\mathcal{G}, \mathcal{P}, \mathcal{M}, C)$ fails. The algorithm finally returns false and reports unicast capacity of 1. However, the network has a unicast capacity of 2 indicated by the two paths in Fig. 3.2(b).

We develop our improved same-layer rewiring to fix the above problem as follows. First, an input x_k should not be blocked from being visited via same-layer rewiring from any input x_i just because it has been visited via same-layer rewiring from another input x_j. Consider the example in Fig. 3.2. If we allow x_3 to be visited via same-layer rewiring from x_5, the algorithm may succeed in finding two LI paths as indicated in Fig. 3.2(b). However, this needs to be done carefully. Consider again the example in Fig. 3.2. If we allow same-layer rewirings from all inputs, then we might run into an infinite loop of going from x_5 to x_3 via same-layer rewiring and going from x_3 to x_5 via same-layer rewiring and so on.

The goal of a same-layer rewiring operation in iteration $k+1$ is to ensure that every input, which allows the algorithm to maintain k LI S-D paths and can further extend the current partial path, has the opportunity of being explored, while ensuring that we do not enter an infinite loop. In this work we achieve this by using a pair of labels of each node.

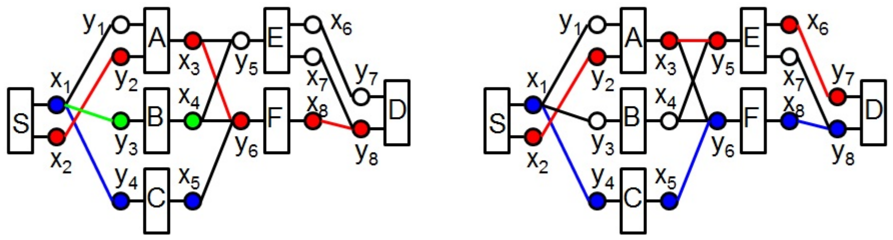


Figure 3.2 Illustrating example for improved same-layer rewiring

Each node has a label that takes values - “explored” or “unexplored”. The other label is a type that takes values 1, 2. We initialize the type of every node to be 1 at the beginning of the iteration. A type 1 input is allowed to initiate same-layer rewirings. An input that is explored via a same-layer rewiring from a type 1 input x_i is assigned as type 2. A type 2 input is not allowed to initiate same-layer rewirings to avoid the possibility of infinite loop. If an input x (of either type) is explored via a backward rewiring, it is re-assigned as type 1 (since U_x^l and U_y^l change since last time x was explored).

Consider applying our improved same-layer rewiring in the example in Fig. 3.2. x_3 is first visited via a same-layer rewiring from x_4 (of type 1) when it is assigned as type 2. Later on x_3 is revisited via a same-layer rewiring from x_5 (of type 1) when it is assigned as type 2 again, so it won’t initiate a same-layer rewiring to x_5, instead it only looks for a possible forward move which happens along the edge (x_3, y_5) (and the improved algorithm finally succeeds in finding 2 LI paths as in Fig. 3.2(b)).

3.3.2 Useful Combinatorial Features

In this subsection, several useful combinatorial features intrinsic in the problem are introduced which are used later in our improved algorithm to reduce the complexity.

In the following, we define a set Λ_{x_i} similar to but more general than L_{x_i} in the original algorithm by Amaudruz and Fragouli. Λ_{x_i} applies to any size of finite field F_p associated with the ADT model for the network.

Definition 1 *Define Λ_{x_i} as a subset of $U_x^{\mathcal{L}(x_i)}$ when x_i is explored such that*

$$T(x_i, U_y^{\mathcal{L}(x_i)}) = \sum_{x_j \in \Lambda_{x_i}} a_{x_i}^j \cdot T(x_j, U_y^{\mathcal{L}(x_i)}). \tag{3.1}$$

where $\{a_{x_i}\}$ are non-zero coefficients from F_p.

Lemma 4 *Λ_{x_i} and the set $\{a_{x_i}\}$ are unique and can be found with complexity $O(k^3)$ in iteration $k+1$.*

Since $T(U_x^{\mathcal{L}(x_i)}, U_y^{\mathcal{L}(x_i)})$ has full-rank, Λ_{x_i} and the set $\{a_{x_i}\}$ are unique and can be found with complexity $O(k^3)$ by using Gaussian elimination.

Let $\mathcal{G}_{x_i}$ denote the bipartite graph containing $U_x^{\mathcal{L}(x_i)} \cup U_y^{\mathcal{L}(x_i)}$ when x_i is explored in iteration $k+1$ and $\mathcal{G}_{x_i}^+$ denote the bipartite graph containing $\{x_i\} \cup U_x^{\mathcal{L}(x_i)} \cup U_y^{\mathcal{L}(x_i)}$.

In the following, we refer to an alternating path as a path in which the edges belong alternatively to the set of used edges and the set of unused edges.

Lemma 5 *There is an alternating path from x_i to any $x_j \in \Lambda_{x_i}$ in the graph $\mathcal{G}_{x_i}^+$ of the form*

$$P_{x_i \to x_j} = \{(x_i, y_1), (x_1, y_1), (x_1, y_2), (x_2, y_2), ...(x_{m-1}, y_m), (x_m = x_j, y_m)\}$$

with $(x_q, y_q), 1 \leq q \leq m$ being edges used by $\mathcal{P}$. The complexity for finding these $|\Lambda_{x_i}|$ paths is bounded by $O(k^2)$ in iteration $k+1$.

Proof

Let $\mathcal{L}(x_i) = l$. Given

$$\text{rank}(T(U_x^l, U_y^l)) = k$$

for any $x_j \in \Lambda_{x_i}$,

$$\text{rank}(T(U_x^l, U_y^l)) = \text{rank}(T(U_x^l + x_i - x_j, U_y^l)) = k$$

where $k = |\mathcal{P}|$ in iteration $k+1$. Introduce an auxiliary output y' and an edge (x_j, y'). It's easy to see that

$$\text{rank}(T(U_x^l + x_i, U_y^l + y')) = k + 1$$

Let $\mathcal{G}_{x_i}^{++}$ denote the bipartite graph containing nodes $\{x_i\} \cup U_x^l \cup U_y^l \cup \{y'\}$.

Given $T(U_x^l, U_y^l)$ has full rank, we know that the polynomial of the determinant of the Edmonds matrix of the bipartite graph $\mathcal{G}_{x_i}$ is not identically zero, so there is a size k perfect matching in $\mathcal{G}_{x_i}$ Motwani & Raghavan (1995), $M_1 = U^l$ giving such a matching. Similarly given

$$\text{rank}(T(U_x^l + x_i, U_y^l + y')) = k + 1$$

there is a size $k+1$ perfect matching in $\mathcal{G}_{x_i}^{++}$. By Berge's Lemma Berge (1957), we know that there is an alternating path, relative to the matching M_1, starting from an unused input x_i to an unused output y', alternating between edges not in the current matching M_1 and edges in the current matching M_1, i.e., there is a path

$$P_{x_i \to y'} = \{(x_i, y_1), (x_1, y_1), (x_1, y_2), (x_2, y_2), ...(x_{m-1}, y_m), (x_m, y_m), (x_m = x_j, y')\}$$

with $(x_q, y_q), 1 \leq q \leq m$ being edges in M_1. So we proved that there is an alternating path

$$P_{x_i \to x_j} = \{(x_i, y_1), (x_1, y_1), (x_1, y_2), (x_2, y_2), ...(x_{m-1}, y_m), (x_m = x_j, y_m)\}$$

with $(x_q, y_q), 1 \leq q \leq m$ being edges in $M_1 = U^l$.

Since the number of nodes in $\mathcal{G}_{x_i}^{+}$ is bounded by $O(k)$, the number of its edges is bounded by $O(k^2)$. Finding $P_{x_i \to x_j}$ for all $x_j \in \Lambda_{x_i}$ in $\mathcal{G}_{x_i}^{+}$ can be done with complexity $O(k^2)$ with some well-known graph traversal algorithms, like breadth-first search Cormen, etc. (2001).

Lemma 6 *Let*

$$\textit{rank}(T(U_x^l, U_y^l)) = |U_x^l| = |U_y^l| = k + 1$$

Given any $y \in U_y^l$, there exists at least one $x \in U_x^l$, such that

$$rank(T(U_x^l - x, U_y^l - y)) = k$$

Moreover there is an alternating path from y to x of the form

$$P_{y\to x} = \{(x_1, y_1 = y), (x_1, y_2), (x_2, y_2), (x_2, y_3), ...(x_{m'-1}, y_{m'}), (x_{m'} = x, y_{m'})\}$$

with $(x_q, y_q), 1 \leq q \leq m'$ being edges in U^l. The complexity of finding one such x is bounded by $O(k^3)$ and the complexity of finding path $P_{y\to x}$ is bounded by $O(k^2)$.

Proof

Given

$$\text{rank}(T(U_x^l, U_y^l)) = |U_x^l| = |U_y^l| = k + 1$$

the determinant of $T(U^l)$, $|T(U^l)|$, is nonzero. Consider the expansion of $|T(U^l)|$ along the column corresponding to y using Laplace expansion, there must be a sub-matrix of $T(U^l)$ with nonzero determinant of size $k \times k$ excluding the column corresponding to y and some row (corresponding to x), i.e.,

$$\text{rank}(T(U_x^l - x, U_y^l - y)) = k$$

Introduce an auxiliary input x' and output y' and edge $(x', y), (x, y')$. Let

$$T_1 = T(U_x^l - x, U_y^l - y)$$

Consider

$$T_2 = T(U_x^l, U_y^l - y + y')$$

Compared with T_1, T_2 has a new column corresponding to y' with a unique non-zero entry on the new row x, so $\text{rank}(T_2) = k + 1$ given that $\text{rank}(T_1) = k$. Consider

$$T_3 = T(U_x^l + x', U_y^l + y')$$

Compared with T_2, T_3 has a new row corresponding to x' with a unique non-zero entry on the new column y, so $\text{rank}(T_3) = k + 2$ given that $\text{rank}(T_2) = k + 1$. Using a similar argument as in Lemma 5, we conclude that there is an alternating path, relative to the matching $M_2 = U^l$, starting from an unused input x' to an unused output y', alternating between edges not in the current matching M_2 and edges in the current matching M_2, i.e., there is a path

$$P_{x' \to y'} = \{(x', y_1 = y), (x_1, y_1), (x_1, y_2), (x_2, y_2), ...(x_{m'-1}, y_{m'}), (x_{m'}, y_{m'}), (x_{m'} = x, y')\}$$

with $(x_q, y_q), 1 \leq q \leq m'$ being edges in M_2. So we proved that there is a path

$$P_{y \to x} = \{(x_1, y_1 = y), (x_1, y_2), (x_2, y_2), (x_2, y_3), ...(x_{m'-1}, y_{m'}), (x_{m'} = x, y_{m'})\}$$

with $(x_q, y_q), 1 \leq q \leq m'$ being edges in U^l.

Finding one such x can be accomplished by performing Gaussian elimination on the matrix $T(U_x^l, U_y^l - y)$ with complexity bounded by $O(k^3)$. Using a similar argument as in Lemma 5, the computational complexity of finding path $P_{y \to x}$ is bounded by $O(k^2)$.

Lemma 7 develops an equivalent but computationally simple method to speed up the rank computation when x_i is explored given Λ_{x_i} and the set of associated coefficients $\{a_{x_i}\}$.

Lemma 7 *Let $T(U_x^l, U_y^l)$ have full rank k. The rank computation for checking rank$(T(U_x^l + x_i, U_y^l + y)) = k$ or $k+1$ for any $x_i \notin U_x^l$, $\mathcal{L}(x_i) = l$, $y \notin U_y^l$ and $(x_i, y) \in \mathcal{E}$ is equivalent to checking*

$$T(x_i, y) = \text{ or } \neq \sum_{x_j \in \Lambda_{x_i}} a_{x_i}^j \cdot T(x_j, y)$$

with complexity bounded by $O(k)$ given Λ_{x_i} and $\{a_{x_i}\}$.

Proof

Given $T(U_x^l, U_y^l)$ has full rank k,

$$\text{rank}(T(U_x^l + x_i, U_y^l + y)) = k$$

is equivalent to that

$$T(x_i, U_y^l + y) = \sum_{x_j \in \Lambda'_{x_i}} a_{x'_i}^j \cdot T(x_j, U_y^l + y)$$

for some $\Lambda'_{x_i} \subseteq U_x^l$ and $\{a_{x'_i}\}$. Since $\Lambda_{x_i} \subseteq U_x^l$ and the set $\{a_{x_i}\}$ are unique for which

$$T(x_i, U_y^l) = \sum_{x_j \in \Lambda_{x_i}} a_{x_i}^j \cdot T(x_j, U_y^l)$$

holds (by Lemma 4), there must be $\Lambda'_{x_i} = \Lambda_{x_i}$ and $\{a_{x_i}\} = \{a_{x'_i}\}$. This leads to that

$$\operatorname{rank}(T(U_x^l + x_i, U_y^l + y)) = k$$

is equivalent to

$$T(x_i, y) = \sum_{x_j \in \Lambda_{x_i}} a_{x_i}^j \cdot T(x_j, y)$$

Lemma 8 *Let $x' \in \Lambda_{x_i}$. If x' is explored via a same-layer rewiring from x_i,*

$$\Lambda_{x'} = \Lambda_{x_i} + x_i - x'$$

and the set of associated coefficients $\{a_{x'}\}$ can be computed from $\{a_{x_i}\}$ with complexity $O(k)$ in iteration $k+1$.

Proof

Let $\mathcal{L}(x_i) = l$. Note that when x' is explored via a same-layer rewiring from x_i, U_x^l is updated as $U_x^l - x' + x_i$, U_y^l is unchanged and $T(U_x^l - x' + x_i, U_y^l)$ has full rank. Based on definition,

$$T(x_i, U_y^l) = \sum_{x_j \in \Lambda_{x_i} \setminus x'} a_{x_i}^j \cdot T(x_j, U_y^l) + a'_{x_i} \cdot T(x', U_y^l). \tag{3.2}$$

where $\{a_{x_i}\}$ are non-zero coefficients from F_p. So we have

$$T(x', U_y^l) = \sum_{x_j \in \Lambda_{x_i} \setminus x'} \frac{a_{x_i}^j}{a'_{x_i}} \cdot T(x_j, U_y^l) - \frac{1}{a'_{x_i}} \cdot T(x_i, U_y^l). \tag{3.3}$$

Since $T(U_x^l - x' + x_i, U_y^l)$ has full rank, equation (3.3) is the unique way that the row $T(x', U_y^l)$ can be expressed as a linear combination of the rows in this matrix. So we conclude $\Lambda_{x'} = \Lambda_{x_i} + x_i - x'$ and the set of associated coefficients $\{a_{x'}\}$ can be computed from $\{a_{x_i}\}$ with complexity $O(k)$. Note that in iteration $k+1$, $|\Lambda_{x_i}| \leq |U_x^l| = k$.

3.3.3 Reducing the Complexity and the Overall Algorithm

As mentioned before, the computational parts of algorithm Amaudruz & Fragouli (2009) include the FindL (finding L_{x_i}), Match (update U after a same-layer rewiring from x_i) and rank computation functions. Now we explain how the combinatorial features from Section 3.3.2 can be used to further reduce the complexity of the unicast algorithm.

Lemma 4 shows that Λ_{x_i} and the set of associated coefficients $\{a_{x_i}\}$ for any type 1 input x_i can be computed with complexity $O(k^3)$ in iteration $k+1$. Lemma 8 tells that for any type 2 input x', $x' \in \Lambda_{x_i}$, that is explored via a same-layer rewiring from a type 1 input x_i, $\Lambda_{x'}$ and the set of associated coefficients $\{a_{x'}\}$ can be computed with complexity $O(k)$ given Λ_{x_i} and the set of associated coefficients $\{a_{x_i}\}$.

Second, based on Lemma 5, the matching or updating of U after same-layer rewirings from any type 1 input x_i can be done with complexity $O(k^2)$ in iteration $k+1$ as follows. First find all $|\Lambda_{x_i}|$ paths $P_{x_i \to x_j}$, $\forall x_j \in \Lambda_{x_i}$ with complexity $O(k^2)$ for x_i. Let

$$P_{x_i \to x_j} = \{(x_i, y_1), (x_1, y_1), (x_1, y_2), ...(x_{m-1}, y_m), (x_m = x_j, y_m)\} = \{e_1, e_2, ..., e_{2m}\}$$

with $(x_q, y_q), 1 \leq q \leq m$ being edges used by $\mathcal{P}$ for any $x_j \in \Lambda_{x_i}$. Then updating of $U^{\mathcal{L}(x_i)}$ after a same-layer rewiring from x_i to x_j can be done by

$$U^{\mathcal{L}(x_i)} \leftarrow U^{\mathcal{L}(x_i)} + e_1 - e_2 + ... - e_{2m}$$

Third, Lemma 7 tells that the rank computation in a forward move from any x_i (either of type 1 or of type 2), $x_i \notin U_x^l$, $\mathcal{L}(x_i) = l$, for checking

$$\mathrm{rank}(T(U_x^l + x_i, U_y^l + y)) = k \text{ or } k+1$$

for any $y \notin U_y^l$ and $(x_i, y) \in \mathcal{E}$ is equivalent to checking

$$T(x_i, y) = \sum_{x_j \in \Lambda_{x_i}} a_{x_i}^j \cdot T(x_j, y) \text{ or not}$$

with complexity bounded by $O(k)$ given Λ_{x_i} and $\{a_{x_i}\}$ in iteration $k+1$.

Finally, as mentioned before, in our improved backward rewiring from an output y, to find one x with $T(U_x^l - x, U_y^l - y)$ having full rank and to rematch $(U_x^l - x, U_y^l - y)$ can be done with complexity $O(k^3)$ in iteration $k+1$ guaranteed by Lemma 6.

Table 3.1 gives an overall description of our improved unicast algorithm which is implemented in a function $E_A(\mathcal{G}, \mathcal{P}, \mathcal{M}, A)$ where all inputs are the same as in the original algorithm. A complete software implementation of our improved unicast algorithm can be found in the author's homepage – http://www.ece.iastate.edu/~cshi.

3.4 Proof of Correctness

Theorem 9 *Our algorithm terminates in finite time. The total computational complexity of our algorithm is bounded by*

$$O(|\mathcal{V}_x|C^4 + d|\mathcal{V}_x|C^3)$$

if our algorithm stops after finding C linearly independent S-D paths in the network, where $|\mathcal{V}_x|$ is the total number of inputs, C is the unicast capacity and d is the maximum number of outgoing edges from any input.

Proof

Our algorithm runs in iterations and each iteration of the algorithm consists of exploring some nodes in the network by visiting some subset of the inputs and outputs of each node. We prove that our algorithm terminates in finite time by proving that it stops in finite iterations and that in each iteration it explores a finite number of inputs and outputs. In Theorem 11 we prove that our algorithm stops after C iterations. Here we prove the total number of inputs/outputs being visited in each iteration of the algorithm is finite.

Note that only unexplored node or input/output may be explored by the algorithm and that once a node or input/output is explored, it's labeled as explored (by $\mathcal{M}$) and not allowed to be explored again unless it is relabeled as unexplored again. Let k_1 and k_2 be the total number of inputs being labeled as unexplored type 1 and type 2 inputs respectively and let k_3 be the total number of outputs being labeled as unexplored outputs which is used by some path found in previous iterations.

Claim 1 $k_3 \leq |\mathcal{V}_x|$.

Table 3.1 Pseudo-code for our improved algorithm

```
${(T,F)}=E_A(\mathcal{G},\mathcal{P},\mathcal{M},A)$
    $\mathcal{M}(A)=T,\mathcal{L}(A)=l$
    $U^l=\{\text{used edges in layer cut } l\}, U^l_x=\{x_i\in U^l\}, U^l_y=\{y_j\in U^l\}$
    for any $x:\mathcal{A}(x)=A, x\notin U^l_x, \mathcal{M}(x)=F, \text{GetType}(x)=2$
        $\mathcal{M}(x)=T$
        for any $y:(x,y)\in\mathcal{E}, y\notin U^l_y, \mathcal{M}(\mathcal{A}(y))=F$ //*forward move*
            if $T(x,y)\neq\sum_{x_j\in\Lambda_x} a^j_x\cdot T(x_j,y)$
                $\text{Update}(\mathcal{P}); U^l\leftarrow U^l+e$
                if $\mathcal{A}(y)=D$, return (T)
                else if $E_A(\mathcal{G},\mathcal{P},\mathcal{M},\mathcal{A}(y))=T$, return(T)
                $U^l\leftarrow U^l-e$; $\text{Restore}(\mathcal{P})$
    for any $x:\mathcal{A}(x)=A, x\notin U^l_x, \mathcal{M}(x)=F, \text{GetType}(x)=1$
        $\mathcal{M}(x)=T$
        Compute $\Lambda_x$ and the set of coefficients $\{a_x\}$
        for any $y:(x,y)\in\mathcal{E}, y\notin U^l_y, \mathcal{M}(\mathcal{A}(y))=F$ //*forward move*
            if $T(x,y)\neq\sum_{x_j\in\Lambda_x} a^j_x\cdot T(x_j,y)$
                $\text{Update}(\mathcal{P}); U^l\leftarrow U^l+e$
                if $\mathcal{A}(y)=D$, return (T)
                else if $E_A(\mathcal{G},\mathcal{P},\mathcal{M},\mathcal{A}(y))=T$, return(T)
                $U^l\leftarrow U^l-e$; $\text{Restore}(\mathcal{P})$
        Find all paths $P_{x\to x_j}$ for all $\forall x_j\in\Lambda_x$
        for any $x_j:x_j\in\Lambda_x$ with $P_{x\to x_j}=\{e_1,e_2,...e_{2m}\}=$
        $\{(x,y_1),(x_1,y_1),(x_1,y_2),...(x_m=x_j,y_m)\}$ //*same-layer rewiring*
            $\mathcal{M}(x_j)=F$; $\text{SetType}(x_j,2)$;
            $\Lambda_{x_j}=\Lambda_x-x_j+x$
            compute $\{a_{x_j}\}$ based on $\{a_x\}$ according to Lemma 8
            $\text{Update}(\mathcal{P}); U^l\leftarrow U^l+e_1-e_2+...+e_{2m-1}-e_{2m}$
            if $E_A(\mathcal{G},\mathcal{P},\mathcal{M},\mathcal{A}(x_j))=T$, return(T)
            $U^l\leftarrow U^l-e_1+e_2-...-e_{2m-1}+e_{2m}$; $\text{Restore}(\mathcal{P})$
    for any $y:\mathcal{A}(y)=A, y\in U^{l-1}_y, \mathcal{M}(y)=F$
    and $y$ is used by $\mathcal{P}$ at the beginning of the iteration //*backward rewiring*
        $\mathcal{M}(y)=T$
        find one $x\in U^{l-1}_x$ with $T(U^{l-1}_x-x, U^{l-1}_y-y)$ having full rank
        and find $P_{y\to x}=\{e_1,e_2,...e_{2m'-1}\}$
        $=\{(x_1,y_1=y),(x_1,y_2),(x_2,y_2),...(x_{m'}=x,y_{m'})\}$
        $\mathcal{M}(x)=F, \text{SetType}(x,1)$
        $\text{Update}(\mathcal{P}); U^{l-1}\leftarrow U^{l-1}-e_1+e_2-...-e_{2m'-1}$
        If $E_A(\mathcal{G},\mathcal{P},\mathcal{M},\mathcal{A}(x))=T$, return (T)
        $U^{l-1}\leftarrow U^{l-1}+e_1-e_2+...+e_{2m'-1}$; $\text{Restore}(\mathcal{P})$
    return (F)
```

Proof

All outputs are labeled unexplored at the beginning of each iteration. After an output is explored, it's labeled as explored and never relabeled as unexplored. Moveover given the total number of outputs used by paths found in previous iterations is no more than $|\mathcal{V}_y|$ and $|\mathcal{V}_x|$, $k_3 \leq |\mathcal{V}_x|$.

Since backward rewiring only happens on unexplored outputs used by paths found in previous iterations and labels the outputs as explored afterwards, the total number of backward rewiring is no more than $k_3 \leq |\mathcal{V}_x|$.

Claim 2 $k_1 \leq 2|\mathcal{V}_x|$.

Proof

All inputs in $\mathcal{V}_x$ are labeled as unexplored type 1 inputs at the beginning of each iteration. An input is relabeled as unexplored type 1 input only after a backward rewiring and we conclude above that the total number of backward rewiring is bounded by $|\mathcal{V}_x|$, so the total number of inputs being relabeled as unexplored type 1 inputs is bounded by $|\mathcal{V}_x|$. Therefore the total number of inputs being labeled as unexplored type 1 inputs is bounded by $2|\mathcal{V}_x|$, that is, $k_1 \leq 2|\mathcal{V}_x|$.

Claim 3 $k_2 \leq 2(k-1)|\mathcal{V}_x|$.

Proof

An input is labeled as unexplored type 2 input only after a same-layer rewiring, so we can prove $k_2 \leq 2(k-1)|\mathcal{V}_x|$ by proving that the total number of same-layer rewirings is bounded by $2(k-1)|\mathcal{V}_x|$. A same-layer rewiring only starts from some type 1 input being explored. The total number of type 1 inputs explored by our algorithm in iteration k is bounded by $k_1 \leq 2|\mathcal{V}_x|$. When a type 1 input x is explored, the total number of same-layer rewirings that starts from x is bounded by $|\Lambda_x| \leq (k-1)$ in iteration k. Therefore the total number of same-layer rewirings is bounded by $2(k-1)|\mathcal{V}_x|$, which leads to $k_2 \leq 2(k-1)|\mathcal{V}_x|$.

The total number of inputs (outputs) explored by our algorithm in iteration k is no more than $k_1 + k_2$ (k_3), which is limited by $2k|\mathcal{V}_x|$ ($|\mathcal{V}_x|$). So we conclude that the total number of inputs/outputs being visited in each iteration of the algorithm is finite. Together with Theorem 11, we proved that our algorithm terminates in finite time.

Now let's consider the computational complexity of our algorithm.

We proved above that in iteration k the total number of type 1, type 2 inputs and outputs explored by our algorithm is bounded by $2|\mathcal{V}_x|$, $2(k-1)|\mathcal{V}_x|$ and $|\mathcal{V}_x|$ respectively. The worst case in computation in iteration k are no more than:

- For each type 1 input x_i, compute Λ_{x_i} and $\{a_{x_i}\}$ with complexity $O(k^3)$ (Lemma 4) and find all paths $P_{x_i \to x_j}$ for $\forall x_j \in \Lambda_{x_i}$ with complexity $O(k^2)$ (Lemma 5),

- For each type 2 input x_j, compute Λ_{x_j} and $\{a_{x_j}\}$ with complexity $O(k)$ (Lemma 8),

- For each type 1 or type 2 input x, compute rank$(T(U_x^l + x, U_y^l + y))$ for any $y \notin U_y^l$, $(x, y) \in \mathcal{E}$ with complexity $O(k)$ given Λ_x and $\{a_x\}$ (Lemma 7) and for any x, the total number of such y is no larger than d and

- In each backward rewiring from a certain y, find one x with $T(U_x^l - x, U_y^l - y)$ having full rank and rematch $(U_x^l - x, U_y^l - y)$ with complexity $O(k^3)$ (Lemma 6). Note that $k \leq C$.

  It's obvious that the total complexity of our improved algorithm is bounded by

$$O(|\mathcal{V}_x|C^4 + d|\mathcal{V}_x|C^3)$$

Table 3.2 lists the comparison results between different algorithms for finding the unicast capacity of linear deterministic wireless relay networks, specially in their complexity.

We note that the issues with the original algorithm Amaudruz & Fragouli (2009) mentioned in Section 3.3.1 have been fixed in Ebrahimi & Fragouli (2009). The main difference between our improved algorithm and the algorithm in Ebrahimi & Fragouli (2009) is that our improved

Table 3.2 Comparison of algorithm complexity

Algorithm	Complexity*	Notes
Amaudruz & Fragouli (2009) Ebrahimi & Fragouli (2009)	$O(M\|\mathcal{E}\|C^5)$ $O(d\|\mathcal{V}_x\|C^5 + \|\mathcal{V}_y\|C^5)$	Always higher than ours especially when C is large
Yazdi & Savari (2009)	$O(L^8M^{12}h_0^3 + LM^6Ch_0^4)$	Always higher than ours, especially when M or L is large
Goemans, etc. (2009)	$O(L^{1.5}M^{3.5}\log(ML))$ or $O(LM^3\log M)$	Straightforward comparison is not possible. Goemans, etc. (2009) will have lower complexity if C is much larger than M
Our work	$O(\|\mathcal{V}_x\|C^4 + d\|\mathcal{V}_x\|C^3)$	-

* Denote C as the unicast capacity, M the maximum number of nodes in each layer, L the total number of layers, d the maximum number of inputs of any node, h_0 the maximum number of inputs/outputs at any layer, E the total number of edges, $|\mathcal{V}_x|$ the total number of inputs and $|\mathcal{V}_y|$ the total number of outputs. Note that $M \geq d$ (since by definition each input can have at most one connection to each node in the next layer), $|\mathcal{E}| \geq |\mathcal{V}_x|$ (because of broadcasting) and $h_0 \geq C$ (based on definition).

algorithm utilizes those useful combinatorial features intrinsic in the problem described in Section 3.3.2 which lead to reduced complexity. The other difference comes from the same-layer rewiring and backward rewiring. In Ebrahimi & Fragouli (2009), the same-layer rewiring starts on each input at most once (using the ML indicator function) while our algorithm allows multiple same-layer rewirings starting from certain inputs (that is, if an input is explored via a backward rewiring, it is reassigned as unexplored type 1 input and allows to initiate same-layer rewiring again). In Ebrahimi & Fragouli (2009), the backward rewiring (implemented in ϕ-function there) allows exploration on every $x_k \in U_x$ such that the resulting adjacency matrix of used path edges still remains full rank while our algorithm only finds one such $x_k \in U_x$ and explores it. Note that it can be verified that the combined effects of the different same-layer rewiring and backward rewiring in two algorithms are the same.

The following lemma is useful in the proof of correctness for the improved algorithm.

Let

$$Y_{x_j,fr}^{K+1} = \{y : y \notin U_y^i \text{ and } \operatorname{rank}(T(U_x^i + x_j, U_y^i + y)) = K + 1$$

when x_j is explored in iteration $K+1\}$ with $\mathcal{L}(x_j) = i$.

Lemma 10 *In the last iteration $K+1$ of our algorithm when no more S-D path is found, all $\mathcal{A}(x_k)$ with $x_k \in \Lambda_{x_j}$ and all $\mathcal{A}(y)$ with $y \in Y^{K+1}_{x_j,fr}$ must have been explored before our algorithm returns if x_j has ever been explored.*

Proof

We first prove that all $\mathcal{A}(x_k)$ with $x_k \in \Lambda_{x_j}$ must have been explored before our algorithm returns if x_j has ever been explored. Since there is no more S-D path exists in iteration $K+1$, all admissible moves will be tried. It follows that if any x_j of type 1 has ever been explored, each $\mathcal{A}(x_k)$ with $x_k \in \Lambda_{x_j}$ will be explored by following a same-layer rewiring along the path $P_{x_j \to x_k}$. For any x_i of type 2 being explored right after a same-layer rewiring along the path $P_{x_j \to x_i}$ starting from some x_j of type 1, we know that

$$\Lambda_{x_i} = \Lambda_{x_j} + x_j - x_i$$

So if all $\mathcal{A}(x_k)$ with $x_k \in \Lambda_{x_j}$ are explored, all $\mathcal{A}(x'_k)$ with $x'_k \in \Lambda_{x_i}$ are also explored.

In the following we prove in two steps that all $\mathcal{A}(y)$ with $y \in Y^{K+1}_{x_j,fr}$ must have been explored before our algorithm returns if x_j has ever been explored. In the first step we prove by contradiction that there must be some $x_l \in \{x_j \cup \Lambda_{x_j}\}$ so that $(x_l, y) \in \mathcal{E}$ for any $y \in Y^{K+1}_{x_j,fr}$. If there is no such x_l, that is, for the column corresponding to y in the matrix $T(U^i_x + x_j, U^i_y + y)$, all entries corresponding to $\{x_j \cup \Lambda_{x_j}\}$ are zeros, so we have

$$T(x_j, U^i_y + y) = \sum_{x_k \in \Lambda_{x_j} \subseteq U^i_x} a^k_{x_j} \cdot T(x_k, U^i_y + y)$$

which leads to

$$\text{rank}(T(U^i_x + x_j, U^i_y + y)) = \text{rank}(T(U^i_x, U^i_y + y)) = K$$

which is a contradiction with the definition for $Y^{K+1}_{x_j,fr}$.

In the second step we prove that the edge (x_l, y) with $x_l \in \{x_j \cup \Lambda_{x_j}\}$ and $y \in Y^{K+1}_{x_j,fr}$ must be considered in a forward move in iteration $K+1$ and then $\mathcal{A}(y)$ be explored given that x_j

has ever been explored. If $x_l = x_j$, (x_j, y) allows a forward move when x_j is explored and then $\mathcal{A}(y)$ will be explored. Now assume $x_l \in \Lambda_{x_j}$. Given x_j is explored, x_l is also explored and

$$U^i_{x_l} + x_l = U^i_{x_j} + x_j$$

and U^i_y is unchanged. Given $y \in Y^{K+1}_{x_j,fr}$, i.e.,

$$\text{rank}(T(U^i_{x_j} + x_j, U^i_y + y)) = K + 1$$

we have

$$\text{rank}(T(U^i_{x_l} + x_l, U^i_y + y)) = K + 1$$

Then (x_l, y) allows a forward move when x_l is explored and then $\mathcal{A}(y)$ will be explored. So all $\mathcal{A}(y)$ with $y \in Y^{K+1}_{x_j,fr}$ must be explored in iteration $K+1$ if x_j has ever been explored.

Theorem 11 *Our algorithm stops after finding C linearly independent S-D paths in a linear layered deterministic relay network $\mathcal{G}$ where C is the unicast capacity of $\mathcal{G}$.*

Proof

Main idea:

First we present the main idea in our proof.

Assume that our algorithm stops after iteration $K + 1$, i.e., our algorithm fails to find an additional S-D path in iteration $K + 1$. We claim that $K = C$.

Those K paths returned by our algorithm correspond to some transmission scheme of rate K from S to D, so we must have $K \leq C$. Next it's sufficient for us to prove that $K \geq C$. We prove $K \geq C$ by proving that when our algorithm stops the number of paths we find, K, equals some cut value in $\mathcal{G}$.

Consider the cut Ω_K separating the nodes labeled explored from the nodes labeled unexplored when the algorithm stops in iteration $K+1$ with S $\in \Omega_K$. Clearly Ω_K is a cut separating S from D and S $\in \Omega_K$, D $\in \Omega^c_K$. We prove $K \geq C$ by proving that this cut value equals K, i.e, $\text{rank}(T(\mathcal{E}_{\Omega_K})) = K$.

Useful notations:

Second we present some useful notations used in our proof.

Let $\mathcal{P}_K$ be the set of S-D paths returned from the first K iterations in our algorithm. Let

$$\mathcal{E}_{\mathcal{P}} = \{(x,y) : (x,y) \in \mathcal{E} \text{ and } (x,y) \text{ is used by } \mathcal{P}_K\}$$

and

$$\mathcal{E}^i_{\mathcal{P}} = \{(x,y) : (x,y) \in \mathcal{E}_{\mathcal{P}} \text{ and } \mathcal{L}(x) = i\}$$

Let

$$\mathcal{V}^i_{x\mathcal{P}} = \{x : (x,y) \in \mathcal{E}^i_{\mathcal{P}}\}$$

and

$$\mathcal{V}^i_{y\mathcal{P}} = \{y : (x,y) \in \mathcal{E}^i_{\mathcal{P}}\}$$

We divide the set $\mathcal{E}^i_{\mathcal{P}}$ into four subgroups:

$$\mathcal{E}^i_{\mathcal{P}1} = \{(x,y) : (x,y) \in \mathcal{E}^i_{\mathcal{P}}, \mathcal{A}(x) \in \Omega_K, \mathcal{A}(y) \in \Omega^c_K\}$$

$$\mathcal{E}^i_{\mathcal{P}2} = \{(x,y) : (x,y) \in \mathcal{E}^i_{\mathcal{P}}, \mathcal{A}(x) \in \Omega_K, \mathcal{A}(y) \in \Omega_K\}$$

$$\mathcal{E}^i_{\mathcal{P}3} = \{(x,y) : (x,y) \in \mathcal{E}^i_{\mathcal{P}}, \mathcal{A}(x) \in \Omega^c_K, \mathcal{A}(y) \in \Omega^c_K\}$$

and

$$\mathcal{E}^i_{\mathcal{P}4} = \{(x,y) : (x,y) \in \mathcal{E}^i_{\mathcal{P}}, \mathcal{A}(x) \in \Omega^c_K, \mathcal{A}(y) \in \Omega_K\}$$

We divide the sets $\mathcal{V}^i_{x\mathcal{P}}$ ($\mathcal{V}^i_{y\mathcal{P}}$) into four subgroups accordingly, $\mathcal{V}^i_{x\mathcal{P}j}$ ($\mathcal{V}^i_{y\mathcal{P}j}$), $1 \leq j \leq 4$. Clearly, the subgroups $\mathcal{V}^i_{x\mathcal{P}j}, 1 \leq j \leq 4$ are disjoint, so are the subgroups $\mathcal{V}^i_{y\mathcal{P}j}, 1 \leq j \leq 4$. Denote

$$|\mathcal{E}^i_{\mathcal{P}1}| = |\mathcal{V}^i_{x\mathcal{P}1}| = |\mathcal{V}^i_{y\mathcal{P}1}| = K_{i1}$$

i.e., K_{i1} is the number of paths (or path edges) in $\mathcal{P}_K$ that cross the cut Ω_K in layer cut i. Denote

$$|\mathcal{E}^i_{\mathcal{P}2}| = K_{i2}$$

$$|\mathcal{E}^i_{\mathcal{P}3}| = K_{i3}$$

and

$$|\mathcal{E}^i_{\mathcal{P}4}| = K_{i4}$$

Since each S-D path crosses each layer cut exactly once,

$$K_{i1} + K_{i2} + K_{i3} + K_{i4} = K, 1 \leq i < L$$

Series of lemmas in the proof:

Lemma 12 says we can prove Theorem 11 or $\text{rank}(T(\mathcal{E}_{\Omega_K})) = K$ by proving that

$$\text{rank}(T^i_{\Omega_K}) = K_{i1} - K_{i4} \text{ for } 1 \leq i < L$$

Lemma 12

$$\sum_{i=1}^{L-1}(K_{i1} - K_{i4}) = \sum_{i=1}^{L-1} K_{i1} - \sum_{i=1}^{L-1} K_{i4} = K \tag{3.4}$$

Proof

Let P be any path in $\mathcal{P}_K$. Let k_p be the times P goes from Ω_K to Ω^c_K and k'_P be the times P goes from Ω^c_K to Ω_K. Given $S \in \Omega_K$ and $D \in \Omega^c_K$, it must be true that

$$k_P - k'_P = 1$$

and

$$\sum_{P \in \mathcal{P}_K} k_P - \sum_{P \in \mathcal{P}_K} k_{P'} = \sum_{P \in \mathcal{P}_K} (k_P - k'_P) = |\mathcal{P}_K| = K$$

By definition, the times that $\forall P \in \mathcal{P}_K$ goes from Ω_K to Ω^c_K is counted in $\sum_{i=1}^{L-1} K_{i1}$ and the times that $\forall P \in \mathcal{P}_K$ goes from Ω^c_K to Ω_K is counted in $\sum_{i=1}^{L-1} K_{i4}$, i.e.,

$$\sum_{P \in \mathcal{P}_K} k_P - \sum_{P \in \mathcal{P}_K} k_{P'} = \sum_{i=1}^{L-1} K_{i1} - \sum_{i=1}^{L-1} K_{i4}$$

So we have

$$\sum_{i=1}^{L-1} K_{i1} - \sum_{i=1}^{L-1} K_{i4} = K$$

Let

$$\mathcal{E}^i_{\Omega_K} = \{(x,y) : (x,y) \in \mathcal{E}, \mathcal{A}(x) \in \Omega_K, \mathcal{A}(y) \in \Omega^c_K \text{ and } \mathcal{L}(x) = i\}, 1 \leq i < L$$

i.e., $\mathcal{E}^i_{\Omega_K}$ is the intersection of the cut Ω_K and layer cut i. Let

$$\mathcal{V}^i_{x\Omega_K} = \{x : (x,y) \in \mathcal{E}^i_{\Omega_K}\}$$

and

$$\mathcal{V}^i_{y\Omega_K} = \{y : (x,y) \in \mathcal{E}^i_{\Omega_K}\}$$

Consider the adjacency matrix $T(\mathcal{E}_{\Omega_K})$ for Ω_K. It is a block diagonal matrix with each sub-block $T^i_{\Omega_K}$ being the adjacency matrix for $\mathcal{E}^i_{\Omega_K}$ and

$$\text{rank}(T(\mathcal{E}_{\Omega_K})) = \sum_{i=1}^{L-1} \text{rank}(T^i_{\Omega_K}) = \sum_{i=1}^{L-1} \text{rank}(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K}))$$

In the rest of the proof, we prove

$$\text{rank}(T(\mathcal{E}_{\Omega_K})) = \sum_{i=1}^{L-1} \text{rank}(T^i_{\Omega_K}) = K$$

by proving

$$\text{rank}(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K})) = K_{i1} - K_{i4}, 1 \leq i < L$$

(based on Lemma 12).

Let

$$\mathcal{V}^i_{x\Omega'_K} = \{x : \mathcal{A}(x) \in \Omega_K \text{ and } \mathcal{L}(x) = i\} \text{ and } \mathcal{V}^i_{y\Omega'_K} = \{y : \mathcal{A}(y) \in \Omega^c_K \text{ and } \mathcal{L}(y) = i+1\}$$

Lemma 13

$$\textit{rank}(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K})) = \textit{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K}))$$

Proof

Based on definition, we have

$$\mathcal{V}^i_{x\Omega'_K} = \mathcal{V}^i_{x\Omega_K} \cup \mathcal{V}^i_{x\Omega''_K}$$

where

$$\mathcal{V}^i_{x\Omega''_K} = \{x : \mathcal{A}(x) \in \Omega_K, \mathcal{L}(x) = i \text{ and there is no } (x, y) \in \mathcal{E}, \forall \mathcal{A}(y) \in \Omega^c_K\}$$

and

$$\mathcal{V}^i_{y\Omega'_K} = \mathcal{V}^i_{y\Omega_K} \cup \mathcal{V}^i_{y\Omega''_K}$$

where

$$\mathcal{V}^i_{y\Omega''_K} = \{y : \mathcal{A}(y) \in \Omega^c_K, \mathcal{L}(y) = i + 1 \text{ and there is no } (x, y) \in \mathcal{E}, \forall \mathcal{A}(x) \in \Omega_K\}$$

So in matrix $T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})$, the rows corresponding to $\mathcal{V}^i_{x\Omega''_K}$ and the columns corresponding to $\mathcal{V}^i_{y\Omega''_K}$ have all-zero entries. So

$$\text{rank}(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K})) = \text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K}))$$

Based on Lemma 13, to prove

$$\text{rank}(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K})) = K_{i1} - K_{i4}, 1 \leq i < L$$

is equivalent to proving

$$\text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}, 1 \leq i < L$$

All the following lemmas, Lemma 14 through 21, contribute to proving

$$\text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}, 1 \leq i < L$$

We introduce more notations that are useful in our proof. Let

$$\mathcal{V}^i_{x\Omega'_K} = \mathcal{V}^i_{x\Omega'_K1} + \mathcal{V}^i_{x\Omega'_K2} + \mathcal{V}^i_{x\Omega'_K3} = \mathcal{V}^i_{x\Omega'_K\{1,2,3\}}$$

(we shall use similar shorthand notations for group union in the following) with

$$\mathcal{V}^i_{x\Omega'_K1} = \mathcal{V}^i_{x\mathcal{P}1}, \mathcal{V}^i_{x\Omega'_K2} = \mathcal{V}^i_{x\mathcal{P}2} \text{ and } \mathcal{V}^i_{x\Omega'_K3} = \mathcal{V}^i_{x\Omega'_K} - \mathcal{V}^i_{x\Omega'_K1} - \mathcal{V}^i_{x\Omega'_K2}$$

Similarly, let

$$\mathcal{V}^i_{y\Omega'_K} = \mathcal{V}^i_{y\Omega'_K\{1,2,3\}}$$

with

$$\mathcal{V}^i_{y\Omega'_K 1} = \mathcal{V}^i_{y\mathcal{P}1}, \mathcal{V}^i_{y\Omega'_K 2} = \mathcal{V}^i_{y\mathcal{P}3} \text{ and } \mathcal{V}^i_{y\Omega'_K 3} = \mathcal{V}^i_{y\Omega'_K} - \mathcal{V}^i_{y\Omega'_K 1} - \mathcal{V}^i_{y\Omega'_K 2}$$

Clearly, the subgroups $\mathcal{V}^i_{x\Omega'_K j}, 1 \leq j \leq 3$ are disjoint, so are the subgroups $\mathcal{V}^i_{y\Omega'_K j}, 1 \leq j \leq 3$.

Moreover we denote U^i as the snapshot of the set of edges in layer cut i being used by paths in $\mathcal{P}$ when a certain input x_k with $\mathcal{L}(x_k) = i$ is being explored in iteration $K+1$ of our algorithm. Here we don't explicitly specify x_k in the notation unless it's necessary to avoid confusion. Let U^i_x and U^i_y be the corresponding sets of inputs and outputs for U^i. Furthermore, divide U^i into $U^i_{\{1,2,3,4\}}$ with

$$U^i_1 = \{(x,y) \in U^i, \mathcal{A}(x) \in \Omega_K, \mathcal{A}(y) \in \Omega^c_K\}$$

$$U^i_2 = \{(x,y) \in U^i, \mathcal{A}(x) \in \Omega_K, \mathcal{A}(y) \in \Omega_K\}$$

$$U^i_3 = \{(x,y) \in U^i, \mathcal{A}(x) \in \Omega^c_K, \mathcal{A}(y) \in \Omega^c_K\}$$

$$U^i_4 = \{(x,y) \in U^i, \mathcal{A}(x) \in \Omega^c_K, \mathcal{A}(y) \in \Omega_K\}$$

Divide U^i_x and U^i_y accordingly into four disjoint subgroups $U^i_{x\{1,2,3,4\}}$ and $U^i_{y\{1,2,3,4\}}$. Based on our algorithm,

$$|U^i_1| + |U^i_2| + |U^i_3| + |U^i_4| = K$$

and $\text{rank}(T(U^i)) = K$. We also have

$$|U^i_j| = |U^i_{xj}| = |U^i_{yj}|, 1 \leq j \leq 4$$

The rest of the proof is organized as follows. From Lemma 14 to 19, we prove that for any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, we have

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) = K_{i1} - K_{i4}, 1 \leq i < L$$

Specifically, we prove it by first proving

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \geq K_{i1} - K_{i4}$$

from Lemma 14 to 16 and second proving

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \leq K_i - K_{i4}$$

from Lemma 17 to 19. Then from Lemma 20 to 21 we prove that when we extend $U^i_{x\{1,2\}}$ to $\mathcal{V}^i_{x\Omega'_K}$ and extend $U^i_{y\{1,3\}}$ to $\mathcal{V}^i_{y\Omega'_K}$, the resultant matrix still have the same rank, i.e.,

$$\text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}, 1 \leq i < L$$

Lemma 14 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, we have $\Lambda_{x_k} \subseteq U^i_{x\{1,2\}}$ and*

$$\textit{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_y)) = \textit{rank}(T(U^i_{x\{1,2\}}, U^i_y)) = |U^i_{x1}| + |U^i_{x2}| \tag{3.5}$$

For any $y \in \mathcal{V}^i_{y\Omega'_K 3}$, we have

$$\textit{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_y + y)) = \textit{rank}(T(U^i_{x\{1,2\}}, U^i_y + y)) = |U^i_{x1}| + |U^i_{x2}| \tag{3.6}$$

Proof

Based on Lemma 10, all $\mathcal{A}(x_j)$ with $x_j \in \Lambda_{x_k}$ will finally be explored in iteration $K+1$ if x_k has ever been explored. By definition, $\mathcal{A}(x)$ is not explored for any $x \in U^i_{x\{3,4\}}$, so we have $\Lambda_{x_k} \subseteq U^i_{x\{1,2\}}$. By definition of Λ_{x_k}, it's easy to conclude that (3.5) holds.

By definition $\mathcal{A}(y)$ is not explored for any given $y \in \mathcal{V}^i_{y\Omega'_K 3}$. Based on Lemma 10, we must have $y \notin Y^{K+1}_{x_k,fr}$, i.e.,

$$\text{rank}(T(U^i_x + x_k, U^i_y + y)) = K$$

Given

$$\text{rank}(T(U^i_x, U^i_y + y)) = K$$

we have

$$T(x_k, U^i_y + y) = \sum_{x_j \in \Lambda'} {a^j_{x_k}}' \cdot T(x_j, U^i_y + y) \text{ for some } \Lambda' \subseteq U^i_x$$

From Definition 1, $\Lambda_{x_k} \subseteq U^i_x$ and $\{a^j_{x_k}\}$ are the unique sets satisfying

$$T(x_k, U^i_y)) = \sum_{x_j \in \Lambda_{x_k}} a^j_{x_k} \cdot T(x_j, U^i_y)$$

so we conclude $\Lambda' = \Lambda_{x_k}$ and ${a^j_{x_k}}' = a^j_{x_k}$, i.e.,

$$T(x_k, U^i_y + y) = \sum_{x_j \in \Lambda_{x_k}} a^j_{x_k} \cdot T(x_j, U^i_y + y)$$

We already proved $\Lambda_{x_k} \subseteq U^i_{x\{1,2\}}$, so

$$\mathrm{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_y + y)) = \mathrm{rank}(T(U^i_{x\{1,2\}}, U^i_y + y)) = |U^i_{x1}| + |U^i_{x2}|$$

Lemma 15 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, we have*

$$U^i_{x\{3,4\}} = \mathcal{V}^i_{x\mathcal{P}\{3,4\}} \text{ and } U^i_{y\{1,3\}} = \mathcal{V}^i_{y\mathcal{P}\{1,3\}}$$

which leads to

$$|U^i_{x1}| + |U^i_{x3}| = |\mathcal{V}^i_{x\mathcal{P}1}| + |\mathcal{V}^i_{x\mathcal{P}3}| = K_{i1} + K_{i3}$$

$$|U^i_{x3}| + |U^i_{x4}| = |\mathcal{V}^i_{x\mathcal{P}3}| + |\mathcal{V}^i_{x\mathcal{P}4}| = K_{i3} + K_{i4}$$

and

$$|U^i_{x1}| - |U^i_{x4}| = |\mathcal{V}^i_{x\mathcal{P}1}| - |\mathcal{V}^i_{x\mathcal{P}4}| = K_{i1} - K_{i4}$$

Proof

By definition, all nodes $\mathcal{A}(x)$ or $\mathcal{A}(y)$ with $x \in \mathcal{V}^i_{x\mathcal{P}\{3,4\}}$ or $y \in \mathcal{V}^i_{y\mathcal{P}\{1,3\}}$ are labeled unexplored when our algorithm returns, so during the running time of our algorithm in iteration $K+1$, any node $x \in \mathcal{V}^i_{x\mathcal{P}\{3,4\}}$ or $y \in \mathcal{V}^i_{y\mathcal{P}\{1,3\}}$ is never explored and $\mathcal{V}^i_{x\mathcal{P}\{3,4\}} \subseteq U^i_x$ and $\mathcal{V}^i_{y\mathcal{P}\{1,3\}} \subseteq U^i_y$ always hold for any x_k explored with $\mathcal{L}(x_k) = i$.

By definition, any $\mathcal{A}(x)$ (or $\mathcal{A}(y)$) with input x (or output y) used by U^i but not by $\mathcal{E}^i_{\mathcal{P}}$ must be explored, which means $\mathcal{V}^i_{x\mathcal{P}\{3,4\}} \subseteq U^i_x$ is the complete subset of U^i_x satisfying for each x in this set, $\mathcal{A}(x)$ is unexplored and $\mathcal{V}^i_{y\mathcal{P}\{1,3\}} \subseteq U^i_y$ is the complete subset of U^i_y satisfying for each node y in this set, $\mathcal{A}(y)$ is unexplored. So we conclude

$$U^i_{x\{3,4\}} = \mathcal{V}^i_{x\mathcal{P}\{3,4\}} \text{ and } U^i_{y\{1,3\}} = \mathcal{V}^i_{y\mathcal{P}\{1,3\}}$$

Given all subsets of U^i_x or U^i_y are disjoint and $|U^i_j| = |U^i_{xj}| = |U^i_{yj}|, 1 \leq j \leq 4$, we conclude that

$$|U^i_{x1}| + |U^i_{x3}| = |\mathcal{V}^i_{x\mathcal{P}1}| + |\mathcal{V}^i_{x\mathcal{P}3}| = K_{i1} + K_{i3}$$

$$|U^i_{x3}| + |U^i_{x4}| = |\mathcal{V}^i_{x\mathcal{P}3}| + |\mathcal{V}^i_{x\mathcal{P}4}| = K_{i3} + K_{i4}$$

and

$$|U^i_{x1}| - |U^i_{x4}| = |\mathcal{V}^i_{x\mathcal{P}1}| - |\mathcal{V}^i_{x\mathcal{P}4}| = K_{i1} - K_{i4}$$

In the following, we directly apply the result

$$|U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

from Lemma 15.

Lemma 16 proves

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \geq K_{i1} - K_{i4}$$

by contradiction.

Lemma 16 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, we have*

$$rank(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \geq |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Proof

If $K_{i1} - K_{i4} < 0$, the statement is obviously true. Assume $K_{i1} - K_{i4} \geq 0$. We know

$$\text{rank}(T(U^i)) = K = |U^i_1| + |U^i_2| + |U^i_3| + |U^i_4|$$

Assume that

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) < |U^i_{x1}| - |U^i_{x4}|$$

then we would have

$$\begin{aligned}
\text{rank}(T(U^i)) &= \text{rank}(T(U^i_{x\{1,2,3,4\}}, U^i_{y\{1,2,3,4\}})) \\
&\leq \text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) + |U^i_{x\{3,4\}}| + |U^i_{y\{2,4\}}| \\
&< |U^i_{x1}| - |U^i_{x4}| + |U^i_{x3}| + |U^i_{x4}| + |U^i_{x2}| + |U^i_{x4}| \\
&= K
\end{aligned}$$

which is a contradiction.

Lemma 17 through 19 proves

$$\mathrm{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \leq K_{i1} - K_{i4}$$

by contradiction.

Let $y \in U^i_{y\{2,4\}}$. By definition, $\mathcal{A}(y)$ is explored in iteration $K+1$. If $y \in \mathcal{V}^i_{y\mathcal{P}}$, then according to our algorithm y must have been deleted from U^i_y in a backward rewiring along $P_{y\to x}$ for some x in iteration $K+1$ when $\mathcal{A}(y)$ is explored. This backward rewiring may happen either after the current exploration of x_k or before the current exploration of x_k in which case y must have already been added back to U^i_y before the current exploration of x_k, otherwise it won't appear in U^i_y when x_k is explored. If $y \notin \mathcal{V}^i_{y\mathcal{P}}$, then according to our algorithm y must have been added to U^i_y in a forward move along edge (x, y) for some x before the current exploration to x_k otherwise y won't appear in U^i_y when x_k is explored. It means for each $y \in U^i_{y\{2,4\}}$, y is either added to U^i_y before the current exploration of x_k or is deleted from U^i_y after the current exploration of x_k.

Lemma 17 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, if*

$$rank(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

then there exists some nonempty set $\mathcal{V}^i_{y24} \subseteq U^i_{y\{2,4\}}$, such that for any $y \in \mathcal{V}^i_{y24}$,

$$T(U^i_{x\{1,2\}}, y) = \sum_{y_j \in \mathcal{V}'_y} a^j_y \cdot T(U^i_{x\{1,2\}}, y_j)$$

for some $\mathcal{V}'_y = \mathcal{V}^i_{y24} - y + \mathcal{V}''_y$ with $\mathcal{V}''_y \subseteq U^i_{y\{1,3\}}$.

Proof

Given

$$\mathrm{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,2,3,4\}})) = |U^i_{x1}| + |U^i_{x2}|$$

if

$$\mathrm{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}}) > |U^i_{x1}| - |U^i_{x4}|$$

we claim there must exist some $y' \in U^i_{y\{2,4\}}$, such that

$$T(U^i_{x\{1,2\}}, y') = \sum_{y_j \in \mathcal{V}_{y'}} a^j_{y'} \cdot T(U^i_{x\{1,2\}}, y_j) \text{ for some } \mathcal{V}_{y'} \subseteq U^i_{y\{1,2,3,4\}} - y'$$

We prove this claim by contradiction. Assume for any $y' \in U^i_{y\{2,4\}}$,

$$T(U^i_{x\{1,2\}}, y') \neq \sum_{y_j \in \mathcal{V}_{y'}} a^j_{y'} \cdot T(U^i_{x\{1,2\}}, y_j) \text{ for any } \mathcal{V}_{y'} \subseteq U^i_{y\{1,2,3,4\}} - y'$$

then we would have

$$\begin{aligned} \text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,2,3,4\}})) &= \text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) + |U^i_{y2}| + |U^i_{y4}| \\ &> |U^i_{x1}| - |U^i_{x4}| + |U^i_{x2}| + |U^i_{x4}| \\ &= |U^i_{x1}| + |U^i_{x2}| \end{aligned}$$

which is a contradiction. So the above claim holds. Now let

$$\mathcal{V}^i_{y24} = \{y' + \mathcal{V}_{y'}\} \cap U^i_{y\{2,4\}}$$

Obviously $\mathcal{V}^i_{y24} \neq \emptyset$ (with at least y'). Let

$$\mathcal{V}''_y = y' + \mathcal{V}_{y'} - \mathcal{V}^i_{y24} \subseteq U^i_{y\{1,3\}}$$

Then

$$\sum_{y_j \in \mathcal{V}^i_{y24}} a^j_{y'} \cdot T(U^i_{x\{1,2\}}, y_j) = \sum_{y_j \in \mathcal{V}''_y} a^j_{y''} \cdot T(U^i_{x\{1,2\}}, y_j)$$

So for any $y \in \mathcal{V}^i_{y24}$, we have

$$\begin{aligned} T(U^i_{x\{1,2\}}, y) &= \sum_{y_j \in \mathcal{V}^i_{y24} - y} a^j_y \cdot T(U^i_{x\{1,2\}}, y_j) + \sum_{y_j \in \mathcal{V}''_y} a^{j}_y{''} \cdot T(U^i_{x\{1,2\}}, y_j) \\ &= \sum_{y_j \in \mathcal{V}'_y} a^j_y \cdot T(U^i_{x\{1,2\}}, y_j) \end{aligned}$$

with $\mathcal{V}'_y = \mathcal{V}^i_{y24} - y + \mathcal{V}''_y$.

Lemma 17 says if

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

then there is a nonempty $\mathcal{V}^i_{y24}$ as described in Lemma 17. Now assume $\mathrm{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > K_{i1} - K_{i4}$ and let $y' \in \mathcal{V}^i_{y24} \subseteq U^i_{y\{2,4\}}$ be the last one in $\mathcal{V}^i_{y24}$ being added to the set U^i_y before the current exploration of x_k or the first one in $\mathcal{V}^i_{y24}$ being deleted from the set U^i_y after the current exploration of x_k. By definition of $\mathcal{V}^i_{y24}$,

$$T(U^i_{x\{1,2\}}, y') = \sum_{y_j \in \mathcal{V}'_y} a^j_y \cdot T(U^i_{x\{1,2\}}, y_j) \tag{3.7}$$

for some $\mathcal{V}'_y = \mathcal{V}^i_{y24} - y' + \mathcal{V}''_y \subseteq U^i_y - y'$ with $\mathcal{V}''_y \subseteq U^i_{y\{1,3\}}$. Let x' be the corresponding input being added or deleted with y'.

Lemma 18 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, assume*

$$rank(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

and let y', x' be defined above. Then just after adding y' or just before deleting y', we have

$$T(U^i_{x\{1,2\}} + x', y') = \sum_{y_j \in \mathcal{V}'_y} a^j_y \cdot T(U^i_{x\{1,2\}} + x', y_j) \tag{3.8}$$

for the same $\mathcal{V}'_y$ as in Equation (3.7). And when x' is explored just before adding y' or just after deleting y', we have

$$rank(T(U^i_{x\{1,2\}} + x', U^i_y)) = |U^i_{x1}| + |U^i_{x2}| + 1 \tag{3.9}$$

Note that in equations (3.8) and (3.9), $U^i_{xj}, U^i_{yj}, 1 \le j \le 4$ represent the corresponding sets when x' is being explored.

Proof

Since $y' \in \mathcal{V}^i_{y24}$ is the last one in the set $\mathcal{V}^i_{y24}$ being added to the set U^i_y before the current exploration of x_k or the first one in the set $\mathcal{V}^i_{y24}$ being deleted from the set U^i_y after the current exploration of x_k, so $\mathcal{V}'_y$ in equation (3.8) is the same as in (3.7) (given also $\mathcal{V}''_y \subseteq (U^i_{y\{1,3\}} = \mathcal{V}^i_{y\mathcal{P}\{1,3\}})$ is not changed), but the set $U^i_{x\{1,2\}}$ in equation (3.7) are changing to $U^i_{x\{1,2\}} + x'$ in equation (3.8). When our algorithm proceeds from the point of just after adding y', x' to U^i_y, U^i_x to the point of exploring x_k or from the point of exploring x_k to the point of just before deleting y', x' from U_y^i, U_x^i, only three kinds of moves are allowed, i.e., the forward move, the same-layer rewiring and the backward rewiring. It is sufficient for us to show that any such move or the backtracking of such move doesn't change the linear relationship among the columns in equation (3.7) so that equation (3.8) holds.

Let's first consider the three moves. A forward move along edge (x, y) would change U_x^i to $U_x^i + x$. Since $\Lambda_x \subseteq U_{x\{1,2\}}^i$ (based on Lemma 14), the vector $T(x, U_y^i)$ is a linear combination of row vectors in $T(U_{x\{1,2\}}^i, U_y^i)$, so the relationship in (3.7) still holds when $U_{x\{1,2\}}^i$ changes to $U_{x\{1,2\}}^i + x$ in a forward move. In a same-layer rewiring, $U_{x\{1,2\}}^i$ changes to $U_{x\{1,2\}}^i + x - x_j$ for some $x_j \in \Lambda_x \subseteq U_{x\{1,2\}}^i$ (based on Lemma 14). Again the vector $T(x, U_y^i)$ is a linear combination of row vectors in $T(U_{x\{1,2\}}^i, U_y^i)$, so the relationship in (3.7) still holds when $U_{x\{1,2\}}^i$ changes to $U_{x\{1,2\}}^i + x - x_j$. In a backward rewiring along $P_{y\to x}$, some $y \in U_y^i - \mathcal{V}_{y24}^i$ is deleted and some $x \in U_{x\{1,2\}}^i$ is deleted which obviously doesn't affect the relationship in (3.7).

Now let's consider backtracking of these moves. Let equation (3.7) hold after a backward rewiring along $P_{y\to x}$ for some $y \notin \mathcal{V}_{y24}^i$. After this move, our algorithm will explore $\mathcal{A}(x)$ when x will be explored with $\Lambda_x \subseteq U_{x\{1,2\}}^i$ (based on Lemma 14), so equation (3.7) should hold before the backward move when $U_{x\{1,2\}}^i$ was $U_{x\{1,2\}}^i + x$. A proceeding same-layer rewiring before the current exploration of x means $U_{x\{1,2\}}^i$ was $U_{x\{1,2\}}^i + x - x'$ with $x \in \Lambda_{x'}$ before the move. Let equation (3.7) hold after this move when x is explored. Again $\Lambda_x \subseteq U_{x\{1,2\}}^i$ (based on Lemma 14), so equation (3.7) holds with the addition of the row for x, which means equation (3.7) holds before the same-layer move when $U_{x\{1,2\}}^i$ was $U_{x\{1,2\}}^i + x - x'$. If the proceeding move is a forward move, it means that $U_{x\{1,2\}}^i$ was $U_{x\{1,2\}}^i - x$. It is obvious that equation (3.7) holds before this move with rows in $U_{x\{1,2\}}^i - x$ if it holds after this move with rows in $U_{x\{1,2\}}^i$. From above discussion, we conclude that equation (3.8) holds.

After adding y' or before deleting y' from U_y^i, we have

$$\operatorname{rank}(T(U_{x\{1,2\}}^i + x', U_{yu}^i + y')) = |U_{x1}^i| + |U_{x2}^i| + 1$$

Given that equation (3.8) holds, we have

$$\operatorname{rank}(T(U_{x\{1,2\}}^i + x', U_y^i)) = |U_{x1}^i| + |U_{x2}^i| + 1$$

when x' is explored just before adding y' or just after deleting y'.

Lemma 19 *For any input x_k with $\mathcal{L}(x_k) = i$, when it is explored in iteration $K+1$, we have*

$$\mathit{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) = |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4} \geq 0$$

Proof

First we prove $K_{i1} - K_{i4} \geq 0$ by contradiction. Assume $K_{i1} - K_{i4} < 0$. Then we must have

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Based on Lemma 17 and 18, if

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

then we will have

$$\text{rank}(T(U^i_{x\{1,2\}} + x', U^i_{y})) = |U^i_{x,1}| + |U^i_{x2}| + 1$$

when x' is explored just before adding y' or just after deleting y' for x', y' defined as in Lemma 18, but it is a contradiction with Lemma 14, so we must have $K_{i1} - K_{i4} \geq 0$.

Second we prove

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) = K_{i1} - K_{i4}$$

with $K_{i1} - K_{i4} \geq 0$. Assume

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) > |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Using a similar argument as above, we would arrive at a contradiction. So we must have

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \leq |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Now together with Lemma 16, we conclude that

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) = |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Lemma 20 *For any input x_k with $\mathcal{L}(x_k) = i$ and any $y_j \in \mathcal{V}^i_{y\Omega'_K 3}$, when x_k is explored in iteration $K+1$, we have*

$$\begin{aligned} rank(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + y_j)) &= rank(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}})) \\ &= |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4} \end{aligned}$$

Proof

First we prove

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}})) = K_{i1} - K_{i4}$$

Based on Lemma 14, $\Lambda_{x_k} \subseteq U^i_{x\{1,2\}}$, so

$$\begin{aligned} \text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}})) &= \text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}})) \\ &= |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4} \end{aligned}$$

(the second equality follows Lemma 19).

Second we prove

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + y_j)) = |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Given

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}})) = |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + y_j))$ equals either $|U^i_{x1}| - |U^i_{x4}|$ or $|U^i_{x1}| - |U^i_{x4}| + 1$. Assume

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + y_j)) = |U^i_{x1}| - |U^i_{x4}| + 1 \tag{3.10}$$

From Lemma 14, we have (3.11) and (3.12),

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_y + y_j)) = \text{rank}(T(U^i_{x\{1,2\}}, U^i_y + y_j)) = |U^i_{x1}| + |U^i_{x2}| \tag{3.11}$$

$$T(x_k, U^i_y + y_j) = \sum_{x_m \in \Lambda_{x_k} \subseteq U^i_{x\{1,2\}}} a^m_{x_k} \cdot T(x_m, U^i_y + y_j) \tag{3.12}$$

From (3.10) and (3.12), we have

$$\text{rank}(T(U^i_{x\{1,2\}}, U^i_{y\{1,3\}} + y_j)) = |U^i_{x1}| - |U^i_{x4}| + 1 \tag{3.13}$$

From (3.11) and (3.13), we have

$$T(U^i_{x\{1,2\}}, y_j) = \sum_{y' \in \mathcal{V}_{y13}} a^{'}_{y_j} \cdot T(U^i_{x\{1,2\}}, y') + \sum_{y'' \in \mathcal{V}_{y24}} a^{''}_{y_j} \cdot T(U^i_{x\{1,2\}}, y'') \tag{3.14}$$

for some $\mathcal{V}_{y13} \subseteq U^i_{y\{1,3\}}$ and some nonempty $\mathcal{V}_{y24} \subseteq U^i_{y\{2,4\}}$. From (3.12) and (3.14), we have

$$T(U^i_{x\{1,2\}} + x_k, y_j) = \sum_{y' \in \mathcal{V}_{y13}} a^{'}_{y_j} \cdot T(U^i_{x\{1,2\}} + x_k, y') + \sum_{y'' \in \mathcal{V}_{y24}} a^{''}_{y_j} \cdot T(U^i_{x\{1,2\}} + x_k, y'') \tag{3.15}$$

for the same $\mathcal{V}_{y13}$ and $\mathcal{V}_{y24}$ in (3.14).

Let $y' \in \mathcal{V}_{y24}$ be the last one in $\mathcal{V}_{y24}$ being added to the set U^i_y before the current exploration of x_k or the first one in $\mathcal{V}_{y24}$ being deleted from the set U^i_y after the current exploration of x_k and let x' be the corresponding input being added or deleted with y'. Then using a similar argument as in Lemma 18, we have

$$\text{rank}(T(U^i_{x\{1,2\}} + x', U^i_y + y_j)) = |U^i_{x1}| + |U^i_{x2}| + 1$$

when x' is explored just before adding y' or just after deleting y', but it's a contradiction with Lemma 14. So it must be

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + y_j)) = |U^i_{x1}| - |U^i_{x4}| = K_{i1} - K_{i4}$$

Lemma 21

$$\begin{aligned} rank(T^i_{\Omega_K}) &= rank(T(\mathcal{V}^i_{x\Omega_K}, \mathcal{V}^i_{y\Omega_K})) \\ &= K_{i1} - K_{i4}, 1 \le i \le L-1 \end{aligned}$$

Proof

Based on Lemma 13, it's sufficient to prove

$$\text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}, 1 \le i \le L-1$$

If $\mathcal{E}^i_{\Omega_K} = \emptyset$, i.e., the cut Ω_K and layer cut i has no intersection, then

$$\text{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4} = 0$$

holds. Next assume that $\mathcal{E}^i_{\Omega_K} \neq \emptyset$.

From Lemma 14 and Lemma 20, it's easy to conclude that

$$\text{rank}(T(U^i_{x\{1,2\}} + x_k, U^i_{y\{1,3\}} + \mathcal{V}^i_{y\Omega'_K 3} = \mathcal{V}^i_{y\Omega'_K})) = \text{rank}(T(U^i_{x\{1,2\}}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4} \quad (3.16)$$

where $U^i_{y\{1,3\}} = \mathcal{V}^i_{y\mathcal{P}\{1,3\}} = \mathcal{V}^i_{y\Omega'_K\{1,2\}}$ (based on Lemma 15) and

$$T(x_k, \mathcal{V}^i_{y\Omega'_K}) = \sum_{x_m \in \Lambda_{x_k} \subseteq U^i_{x\{1,2\}}} a^m_{x_k} \cdot T(x_m, \mathcal{V}^i_{y\Omega'_K}) \quad (3.17)$$

for any input x_k with $\mathcal{L}(x_k) = i$ explored in iteration $K+1$.

Let $x^i_q, 1 \leq q \leq Q$ be the qth input in layer i that has been explored in iteration $K+1$ in our algorithm. Note that since some inputs may be explored more than once, x^i_q may not be distinct but Q is finite. We claim that for $1 \leq q \leq Q$,

$$\text{rank}(T(\mathcal{V}^i_{x\mathcal{P}\{1,2\}} + \sum_{k=1}^{q} x^i_k, \mathcal{V}^i_{y\Omega'_K})) = \text{rank}(T(\mathcal{V}^i_{x\mathcal{P}\{1,2\}}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4} \quad (3.18)$$

Now we prove (3.18). When x^i_1 is explored, $U^i_{x_1\{1,2\}} = \mathcal{V}^i_{x\mathcal{P}\{1,2\}}$. From (3.16), we have

$$\text{rank}(T(\mathcal{V}^i_{x\mathcal{P}\{1,2\}} + x^i_1, \mathcal{V}^i_{y\Omega'_K})) = \text{rank}(T(\mathcal{V}^i_{x\mathcal{P}\{1,2\}}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4} \quad (3.19)$$

When x^i_2 is explored, $U^i_{x_2\{1,2\}} \subseteq U^i_{x_1\{1,2\}} + x^i_1$, and from (3.16)

$$\text{rank}(T(U^i_{x_2\{1,2\}} + x^i_2, \mathcal{V}^i_{y\Omega'_K})) = \text{rank}(T(U^i_{x_2\{1,2\}}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4} \quad (3.20)$$

From (3.19) and (3.20), we conclude that

$$\text{rank}(T(\mathcal{V}^i_{x\mathcal{P}\{1,2\}} + x^i_1 + x^i_2, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}$$

Use induction on $x^i_q, 1 \leq q \leq Q$, just as we did for x^i_2, we conclude that (3.18) holds for any q,$1 \leq q \leq Q$.

We know that

$$\mathcal{V}^i_{x\mathcal{P}\{1,2\}} + \sum_{k=1}^{Q} x^i_k = \mathcal{V}^i_{x\Omega'_K}$$

so when $q = Q$ equation (3.18) means that

$$\mathrm{rank}(T(\mathcal{V}^i_{x\Omega'_K}, \mathcal{V}^i_{y\Omega'_K})) = K_{i1} - K_{i4}, 1 \leq i \leq L-1$$

From Lemma 21, we have

$$\mathrm{rank}(T(\mathcal{E}_{\Omega_K})) = \sum_{i=1}^{L-1} \mathrm{rank}(T^i_{\Omega_K}) = \sum_{i=1}^{L-1} (K_{i1} - K_{i4})$$

and from Lemma 12, we have

$$\sum_{i=1}^{L-1} (K_{i1} - K_{i4}) = K$$

so we conclude

$$\mathrm{rank}(T(\mathcal{E}_{\Omega_K})) = K$$

This concludes our proof for Theorem 11.

Theorems 9 and 11 prove that our algorithm terminates in finite time. Theorem 11 proves that our algorithm returns C linearly independent S-D paths where C is the unicast capacity of the underlying deterministic relay network. They consist of the complete proof of correctness for our algorithm for finding the unicast capacity of any linear layered deterministic wireless relay network.

3.5 Conclusions

In this chapter we present an improved algorithm for finding the unicast capacity of linear deterministic wireless networks. Our algorithm improves upon the original algorithm by Amaudruz & Fragouli (2009). We amend the original algorithm so that it finds the unicast capacity correctly for any given deterministic networks. Moreover we fully explore several useful combinatorial features intrinsic in the problem which lead to reduced complexity. Our improved algorithm applies with any size of finite fields associated with the ADT model defining the network. Our improved algorithm proves to be very competitive when comparing with other algorithms on solving the same problem in terms of complexity.

CHAPTER 4. DESIGN AND ANALYSIS OF RATE COMPATIBLE LDPC CODES

As introduced in Chapter 1, rate-compatible LDPC codes are a good choice for changing channel conditions, like in wireless communications. The previous work on the design and analysis of LDPC codes are all targeting at a specific code rate and no work is known on the design and analysis of rate-compatible LDPC codes so that the code performance at all code rates in the family are manageable and predictable. In our work, we proposed algorithms for the design and analysis of rate-compatible LDPC codes with good puncturing performance and make the code performance at all code rates manageable and predictable. Our work is based on E^2RC codes, while our approaches in the design and analysis can be applied more generally not only to E^2RC codes, but to other suitable scenarios, like the design of IRA codes (Jin, etc. (2000)). Most encouragingly, we obtain families of rate-compatible codes whose gaps to capacity are at most 0.3 dB across the range of rates when the maximum variable node degree is twenty, which is very promising compared with other existing results.

In this chapter, we consider the design and analysis of the efficiently-encodable rate-compatible (E^2RC) irregular LDPC codes proposed in previous work. In this part we introduce semi-structured E^2RC-like codes and protograph E^2RC codes. EXIT chart based methods are developed for the design of semi-structured E^2RC-like codes that allow us to determine near-optimal degree distributions for the systematic part of the code while taking into account the structure of the deterministic parity part, thus resolving one of the open issues in the original construction. We develop a fast EXIT function computation method that does not rely on Monte-Carlo simulations and can be used in other scenarios as well. Our approach allows us to jointly optimize code performance across the range of rates under puncturing. We then consider protograph E^2RC codes (that have a protograph representation) and propose rules for designing a family of rate-compatible punctured protographs with low thresholds. For both the semi-structured and protograph E^2RC families we obtain codes whose gap to capacity is at most 0.3 dB across the range of rates when the maximum variable node degree is twenty.

4.1 Introduction

Low-density parity-check (LDPC) codes Gallager (1963) have found widespread acceptance in different areas due to their superior performance and low complexity decoding. In this chapter, we investigate rate-compatible punctured LDPC codes that have the flexibility of operating at different code rates while having a single encoder-decoder pair. Rate-compatible punctured codes are defined by specifying a systematic mother code that operates at the lowest code rate. The parity bits of higher rate codes in a rate-compatible code family are subsets of the parity bits of lower rate codes. A number of papers have investigated issues around the design of good rate-compatible punctured LDPC codes. The work of Ha, etc. (2004 - 1) presents methods for finding optimal degree distributions for puncturing. In Ha, etc. (2004 - 2), Ha, etc. (2006), Yue, etc. (2007), algorithms for finding good puncturing patterns for a given mother code were proposed. There have also been attempts to design mother codes (along with puncturing patterns) with good performance under puncturing Kim, etc. (2009), Yazdani & Banihashemi (2004), Kim, etc. (2006).

E^2RC codes introduced in Kim, etc. (2009) are linear-time encodable and have good puncturing performance across a wide range of code rates. In this work we present systematic approaches for the design and analysis of E^2RC-like codes. Let $H = [H_1|H_2]$ denote the parity check matrix of a systematic LDPC code where H_1 denotes the systematic part and H_2 the parity part. We address the design of two types of codes in our work as explained below.

i) *Semi-structured E^2RC-like codes.*

In these codes the parity part H_2 is deterministic. We use the lower triangular form introduced in Kim, etc. (2009) and introduce a protograph structure for the H_2 part. An example is shown in Fig. 4.1. We assume a random edge interleaver between systematic variable nodes and check nodes, which divides the code into a structured part and an unstructured part, as shown in Fig. 4.1. We solve the problem of finding optimal degree distributions for the unstructured part in this case for optimizing the rate-compatible codes at any specified punctured code rate(s).

ii) *Structured E^2RC-like codes.*

These codes are protograph codes as introduced in Thorpe (2003). The distinguishing feature is that the parity part of the protograph has an E^2RC structure. We demonstrate that very good rate-compatible punctured code families can be obtained using the design rules we propose for the protograph construction. The protograph structure is especially valuable in practical applications as it allows parallelized decoding and requires significantly less storage space for the description of the parity-check matrix than unstructured codes when circulant permutations are used.

We obtain semi-structured E^2RC codes that have a small gap to capacity across the range of puncturing rates. Furthermore, we present optimized quasi-cyclic protograph codes based on the E^2RC structure and demonstrate that very good performance can be obtained with them.

This chapter is organized as follows. In Section 4.2, we briefly discuss the main contributions of our work. Section 4.4 presents our new method for the design of semi-structured E^2RC codes. We also discuss the method of predicting the puncturing performance of semi-structured E^2RC codes and the joint optimization of our codes at any specified punctured code rates. We explain the construction of protograph E^2RC codes in Section 4.5, and Section 3.5 outlines our conclusions.

4.2 Main Contributions

We first outline the issues left unresolved in the work of Kim, etc. (2009).

a) The original construction of E^2RC codes proposed the special H_2 (parity part) structure of the parity-check matrix H. However the design of appropriate degree sequences for the H_1 (information part) based on the constrained H_2 structure, was not discussed. In Kim,

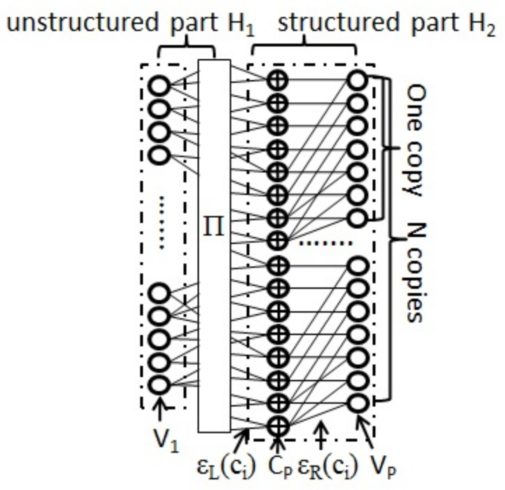


Figure 4.1 Tanner graph representation of E^2RC codes.

etc. (2009), the authors used degree sequences designed for standard irregular codes and constructed H to match these distributions as closely as possible.

b) The construction technique did not provide any means of optimizing code performance at any particular puncturing rate or across all rates simultaneously.

c) As pointed out by an anonymous reviewer, the original E^2RC codes suffer from high error floors Song, etc. (2008) at the mother code rate. As shown in Song, etc. (2008), this is because the H_2 structure causes the maximum check node degree to be large.

d) The original E^2RC codes work with completely random interleavers, that are hard to implement in practice.

In this chapter, we resolve each of the issues discussed above. We briefly overview the main contributions below.

i) *Systematic design techniques for E^2RC-like codes.*

Note that the analysis of E^2RC codes does not follow directly from the analysis of related codes such as systematic IRA codes Jin, etc. (2000), Roumy, etc. (2004). This is because the structured part of IRA codes is symmetric while that of E^2RC codes is quite asymmetric.

In Roumy, etc. (2004), four methods were proposed for the design of IRA codes. The first two methods implicitly assumed one edge type in the accumulator part which was justified by the symmetry of the part. Together with a one-parameter approximation of the message distribution function, Gaussian or BEC approximation, these two methods yielded almost closed-form equations of density evolution. However, one-edge type assumption turns out not accurate enough for the structured part of E^2RC codes because of its asymmetry. In the latter two methods in Roumy, etc. (2004), Monte Carlo simulations were used for generating the EXIT function of the structured part of IRA codes. The Monte Carlo simulation based method is accurate for computing EXIT functions of both symmetric and asymmetric constituent code components by taking the structure of the code component into account. When we design semi-structured E^2RC codes using EXIT chart, we take into account the complete structure of the deterministic part of E^2RC codes to compute the EXIT function as presented in Section 4.4. Instead of resorting to Monte Carlo simulations, we propose a fast and analytical method for computing EXIT functions by solving a set of equations. We use multiple edge types Richardson (2009) for the structured part of E^2RC codes, one edge type for each edge in the protograph representation. So instead of having only three equations (equations (19) (20) (21) in Roumy, etc. (2004)) from the structured part of IRA codes, we have $2|E_R| + |E_L|$ equations from the structured part of E^2RC codes for density evolution. As demonstrated by simulations and the threshold predictions, this introduces a systematic method towards the design of semi-structured E^2RC codes with better performance than the original E^2RC codes.

ii) *A fast technique for EXIT function computation of code components based on protographs.*

Note that usually EXIT functions are computed via Monte-Carlo simulation, which tends to be time-consuming. In this work we present a general technique for computing EXIT functions of code components with a protograph structure. This greatly speeds up the code design process. While we applied it to the design of our semi-structured E^2RC-like codes, it can be applied for any protograph like components, e.g. we can apply it to find the EXIT function of the H_2 part of the IRA code by working with its protograph representation.

iii) *Simultaneous optimization of code performance across multiple rates.*

By exploring the E^2RC structure and its designed puncturing pattern, we propose the design of good rate-compatible punctured codes so that the gap to capacity across the entire range of rates can be controlled. To the best of our knowledge, the current literature does not address this point.

iv) *Alleviating the high error floor problem of the original E^2RC codes.*

In our design of semi-structured E^2RC codes, we impose a protograph structure on the H_2 part, which corresponds to the H_2 part of a very short original E^2RC code. This ensures that the maximum check node degree remains low, thus preventing the high error floors that occur in the original E^2RC codes at mother code rate. For a related approach see Song, etc. (2008).

v) *Design of high-performance codes based on protographs.*

Codes with completely random interleavers are too complex from the point of view of implementation in hardware. In this work, we design protograph E^2RC codes where both the H_1 and the H_2 parts have a protograph structure. We propose design rules for generating a family of rate-compatible protographs with good threshold properties at all punctured rates. Finally, we demonstrate codes with performance better than the original E^2RC codes, that are obtained by replacing the protograph edges by circulant permutations.

4.3 Background and Related Work

An LDPC code can be defined by a parity-check matrix or equivalently by a bipartite (or Tanner) graph representation. For the bipartite graph representation, we follow the convention that a blank circle represents an unpunctured variable node participating in the transmission and a filled circle represents a punctured variable node not participating in the transmission. The asymptotic threshold of LDPC codes can be found by performing density evolution Richardson, etc. (2001), Richardson & Urbanke (2001), Luby, etc. (1997), Shokrollahi (1999) on the degree distribution pair. However, for LDPC codes with structured components such as IRA codes and protograph LDPC codes Thorpe (2003), the density evolution analysis needs to take the underlying structure into account. This can be handled by classifying edges into different types Richardson (2009) and also by using EXIT charts Brink & Kramer (2003). Protograph LDPC codes start with a small mini-graph (called a protograph) and construct the LDPC codes by replacing each edge in the protograph by a random permutation of a fixed size. They can be considered as a subclass of the multi-edge type LDPC codes Richardson (2009). Fast density evolution based on the reciprocal channel approximation Chung (2000) can be performed on protographs to determine their asymptotic threshold.

4.3.1 Efficiently Encodable Rate-Compatible LDPC Codes

We now briefly overview the E^2RC codes introduced in Kim, etc. (2009). Let $H = [H_1|H_2]$ denote the parity-check matrix of an E^2RC code in systematic form. We say that a parity node in H_2 is k-step recoverable (or k-SR) if it can be recovered in exactly k iterations of iterative decoding assuming that all the parity bits are punctured and all the systematic bits are known (Fig. 4.2 shows an example). Intuitively, a large number of low-SR nodes tend to reduce the required number of decoding iterations in the high SNR regime and result in good puncturing performance.

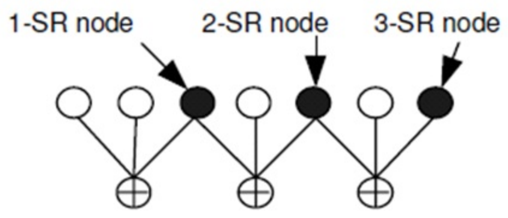


Figure 4.2 The figure shows an example of a 1-SR, 2-SR and 3-SR node.

In Kim, etc. (2009), the submatrix H_2 consists of exclusively degree-2 and degree-1 nodes. Moreover, when the number of parity nodes is a power of two, half the nodes in H_2 are 1-SR, one-fourth are 2-SR and so on. The special structure of H_2 for E^2RC codes allows linear-time encoding and results in good puncturing performance with a puncturing pattern, where 1-SR nodes should be punctured first, 2-SR nodes be punctured next and so on depending upon the rate requirement.

The E^2RC codes have good puncturing performance at relatively short block lengths. However, when the block length gets large, the structure of H_2 may induce a large spread in the check node degree distribution that may cause a loss of performance. In recent work, Song et al. Song, etc. (2008) showed that E^2RC codes exhibit high error floors at their mother code rate and claimed that this stems from their dispersive right degree distribution and high maximum right degree. They presented a modified approach that fixes the high error floor problem. In Section 4.4.2, we show that our approach also effectively eliminates the high error floors of E^2RC codes at the mother code rate. In fact we obtain codes whose performance is slightly better than those in Song, etc. (2008).

4.3.2 Why EXIT Chart

The asymptotic threshold of LDPC codes can be found by performing density evolution Richardson, etc. (2001) Richardson & Urbanke (2001) Luby, etc. (1997) Shokrollahi (1999) on the degree distribution pair. However, for LDPC codes with structured components such as IRA codes and protograph LDPC codes Thorpe (2003), the density evolution analysis needs to take the underlying structure into account. This can be handled by classifying edges into different types Richardson (2009) and also by using EXIT charts Brink & Kramer (2003).

EXIT charts Brink (2001) were first proposed for understanding the convergence behavior of iteratively decoded parallel concatenated codes, and were later generalized to the analysis of LDPC codes Brink & Kramer (2003), Brink, etc. (2004), Ashikhmin, etc. (2004), Sharon, etc. (2006). The components of an EXIT chart are the EXIT functions of the constituent code components of the iterative decoder, which relates the a priori mutual information available to a code component, denoted I_A and the extrinsic mutual information generated after decoding, denoted I_E. The advantage of EXIT charts is that the code design problem can be reduced to a curve fitting problem between the code components (usually two in number).

For log-domain belief propagation decoding of unstructured LDPC codes, if the incoming messages to a variable node v of degree d_v are assumed to be Gaussian and independent, the

EXIT function for the code component involving all variable nodes is given by Brink, etc. (2004)

$$I_{E,V}(I_{A,V}, \sigma^2_{mch}) = \sum_{d_v} \lambda_{d_v} J(\sqrt{(d_v - 1)[J^{-1}(I_{A,V})]^2 + \sigma^2_{mch,v}}) \tag{4.1}$$

where $\sigma^2_{mch,v} = 4/\sigma^2_n$ for unpunctured v (σ^2_n represents the channel noise variance), $\sigma^2_{mch,v} = 0$ for punctured v and $\{\lambda_{d_v}\}$ is the edge perspective degree distribution of variable nodes. Similarly the EXIT function for the code component involving all check nodes is given by

$$I_{E,C}(I_{A,C}) = 1 - \sum_{d_c} \rho_{d_c} J(\sqrt{(d_c - 1)[J^{-1}(1 - I_{A,C})]^2}) \tag{4.2}$$

where $\{\rho_{d_c}\}$ is the edge perspective degree distribution of check nodes.

4.4 Semi-Structured E^2RC-Like Code Design

In this section, we propose our design method for semi-structured E^2RC-like codes using EXIT charts. We consider the unstructured part and the structured part of E^2RC codes shown in Fig. 4.1 as the two constituent code components. This code division for EXIT chart analysis is justified by the random edge interleaver between the two code components.

We denote the set of variable nodes in the Tanner graph by $V = V_1 \cup V_2$ where V_1 is the subset of nodes in H_1 and V_2 is the subset of nodes in H_2. The check node set is denoted by C. In the semi-structured E^2RC codes, H_2 has a base protograph structure of the form proposed in Kim, etc. (2009). The base protograph shall be parameterized by the number of check nodes in it, denoted by M. The H_2 part of semi-structured E^2RC codes is obtained by simply replicating the base protograph an appropriate number of times. For example, the case of $M = 8$ is shown in Fig. 4.1. Check nodes are connected to the set V_1 by a random interleaver (denoted Π in Fig. 4.1). We shall frequently need to refer to the protograph representation of H_2. Let V_p and C_p denote the variable node set and the check node set in the protograph representation of H_2. Let $\varepsilon(v_i)$ and $\varepsilon(c_i)$ denote the set of edges connected to $v_i \in V_p$ and $c_i \in C_p$ respectively. We shall use $\varepsilon_L(c_i)$ to denote the set of edges connecting c_i and the random edge interleaver and use $\varepsilon_R(c_i)$ to denote the set of edges connecting c_i

and V_p, i.e., $\varepsilon(c_i) = \varepsilon_L(c_i) \cup \varepsilon_R(c_i)$. Given a protograph structure on H_2, the problem of code design becomes one of finding good degree distributions for the variable nodes in V_1 and that for the edges in $\cup_{c_i \in C_p} \varepsilon_L(c_i)$ (henceforth referred to as the *left check degree distribution*). In our examples, we only consider concentrated or near-concentrated total check degrees. We have found experimentally that these tend to give the best performance. Note that since the H_2 part is fixed, this implies that the left check degree distribution is also more or less fixed. Accordingly in our design process we experiment with a few check degree distributions and focus on optimizing the degree distribution for the nodes in V_1.

We explain our design method in the context of the binary-input AWGN (BIAWGN) channel. It can be adapted to the BEC and other channels in a straightforward manner. Suppose that we are given a channel noise variance σ_n^2, the protograph specifying H_2 and the left check degree distribution. The code design problem is to find the degree distribution $\{\lambda_{d_v}, v \in V_1\}$ so as to minimize the gap between code rate R and channel capacity C (corresponding to σ_n^2), while constraining the maximum variable node degree to be $d_{v,\max}$. Denote the EXIT function of the structured part by $I_{E,S}(I_{A,S})$. For a given $\{\lambda_{d_v}, v \in V_1\}$, the EXIT function of the unstructured part (see Fig. 4.1) can be expressed as (according to (4.1))

$$I_{E,unS}(I_{A,unS}, \sigma_{mch}^2) = \sum_{d_v} \lambda_{d_v} J(\sqrt{(d_v - 1)[J^{-1}(I_{A,unS})]^2 + \sigma_{mch}^2}) \tag{4.3}$$

The code design or optimization problem is formulated as

$$\begin{aligned} \text{minimize} &: C - R \\ \text{subject to} &: 1.\ \sum_{d_v=1}^{d_{v,max}} \lambda_{d_v} = 1, \lambda_{d_v} \geq 0 \\ & \quad 2.\ I_{E,unS}(I_{A,unS}) > I_{A,S}(I_{E,S}) \\ & \quad\quad \text{for } I_{A,unS} = I_{E,S} \in [0,1) \end{aligned}$$

Here, the second constraint is the zero-error constraint by ensuring the tunnel between the two EXIT curves. It is easy to see that minimizing $C - R$ for a fixed σ_n^2 is equivalent to maximizing $\sum_{d_v=1}^{d_{v,max}} \frac{\lambda_{d_v}}{d_v}$ for $v \in V_1$. The computation of $I_{E,S}(I_{A,S})$ will be elaborated on in Section 4.4.1.

$I_{E,unS}(I_{A,unS})$ is a linear function of $\{\lambda_{d_v}, v \in V_1\}$ as in (4.3). Therefore by a fine enough discretization of the interval $[0, 1)$, we can express the above optimization as a linear program.

In practice, to set up the second constraint, we need to find the inverse map $I_{A,S}(I_{E,S})$ by using linear interpolation. We have found that a large number (say 10^4) of $(I_{A,S}, I_{E,S})$ pairs for the function $I_{E,S}(I_{A,S})$ are necessary to ensure the accuracy of the inverse map $I_{A,S}(I_{E,S})$ and the solution to the optimization problem. By solving the above optimization problem at a certain channel parameter σ_n^2, we get a code of rate corresponding to the $\{\lambda_{d_v}\}$ returned from the optimization. To get an optimized code at rate R_o, we need to solve the above optimization problem at closely spaced channel parameter levels below the Shannon limit corresponding to R_o until we get a code rate close enough to R_o. This necessitates numerous computations of $I_{E,S}(I_{A,S})$ and motivates the need for a fast method for computing $I_{E,S}(I_{A,S})$.

4.4.1 A New Method for Computing EXIT Function of the Structured Part

The usual approach for finding the EXIT function of a constituent code component is proposed in Brink (2001) by using Monte Carlo simulations. A large number of Monte Carlo simulations are needed for obtaining smooth EXIT functions. Moreover, this needs to be repeated at many different channel parameters. This makes the process rather time-consuming.

Here, we present a fast and accurate method for computing EXIT functions of structured code components of LDPC codes, such as the structured part of E^2RC codes and that of IRA codes, without resorting to Monte Carlo simulations.

For convenience, we use the notation

$$E_R = \cup_{v_i \in V_p} \varepsilon(v_i) \text{ and } E_L = \cup_{c_i \in C_p} \varepsilon_L(c_i)$$

Note that $\cup_{v_i \in V_p} \varepsilon(v_i) = \cup_{c_i \in C_p} \varepsilon_R(c_i)$. Suppose that the a priori inputs carried on $e \in E_L$ have average mutual information $I_{A,in}$ and that $v \in V_p$ has channel inputs parameterized by $\sigma^2_{mch,v}$. We are interested in finding I_E, the average mutual information associated with the extrinsic outputs carried on $e \in E_L$ after iterative decoding. For an edge e connected to node $v_i(c_i)$, we shall use the notation $I^{v_i}_{A,e}$(likewise $I^{c_i}_{A,e}$) to denote the mutual information describing the a priori inputs on it and $I^{v_i}_{E,e}$(likewise $I^{c_i}_{E,e}$) the mutual information describing the extrinsic outputs on it. We set up the following system of equations for the given structured code component. For $e \in E_R$ and $v_i \in V_p$, we have

$$I_{E,e}^{v_i} = J\left(\sqrt{\sum_{e' \in \varepsilon(v_i)\setminus\{e\}} [J^{-1}(I_{A,e'}^{v_i})]^2 + \sigma_{mch,v_i}^2}\right). \tag{4.4}$$

Similarly, for $e \in E_R$ and $c_i \in C_p$,

$$I_{E,e}^{c_i} = 1 - J\left(\left(\sum_{e' \in \varepsilon_L(c_i)} [J^{-1}(1 - I_{A,in})]^2 + \sum_{e' \in \varepsilon_R(c_i)\setminus\{e\}} [J^{-1}(1 - I_{A,e'}^{c_i})]^2\right)^{\frac{1}{2}}\right) \tag{4.5}$$

and for $e \in E_L$ and $c_i \in C_p$,

$$I_{E,e}^{c_i} = 1 - J\left(\left(\sum_{e' \in \varepsilon_L(c_i)\setminus\{e\}} [J^{-1}(1 - I_{A,in})]^2 + \sum_{e' \in \varepsilon_R(c_i)} [J^{-1}(1 - I_{A,e'}^{c_i})]^2\right)^{\frac{1}{2}}\right). \tag{4.6}$$

For each edge $e \in E_R$, there are two equations in the form of (4.4) and (4.5) respectively and two unknown variables $I_{E,e}^{v_i}$(or $I_{A,e}^{c_j}$), $I_{E,e}^{c_j}$(or $I_{A,e}^{v_i}$) associated with it; while for each edge $e \in E_L$, there is one equation in the form (4.6) and one unknown variable $I_{E,e}^{c_i}$ associated with it. We want to compute $I_{E,e}^{c_i}$ for $e \in E_L$. There are totally $2|E_R| + |E_L|$ equations and the same number of unknown variables involved in this system of equations. Note that the specific expressions for this system of equations are totally dependent on the structure of the code component. The main idea behind our method for computing the EXIT function is to find the solution to this system of equations for a given value of $I_{A,in}$ and channel parameter. We now present an intuitive method for solving this system of equations, which works in an iterative manner by applying the sequence of updates described in equations (4.4), (4.5) and (4.6). The details are given below.

1) *Problem Instance.*

Given a structured code component, solve the system of equations described in (4.4), (4.5) and (4.6) above. The unknown variables involved in this system of equations are $I_{E,e}^{v_i}$(or $I_{A,e}^{c_j}$), $I_{E,e}^{c_j}$(or $I_{A,e}^{v_i}$) for all $e \in E_R$ and $I_{E,e}^{c_i}$ for all $e \in E_L$. The known variables are $I_{A,e}^{c_i} = I_{A,in}$ for all $e \in E_L$, and the channel parameter σ_n^2 from which σ_{mch,v_i}^2 can be determined for each v_i.

2) *Initialization.*

Initialize all unknown variables to be 0. Set a small value of $\epsilon_{thresh} = 10^{-6}$.

3) *Iterative Updates.*

(a) *Check node update.*

For all $e \in E_R$, compute $I_{E,e}^{c_i}$ using equation (4.5). Check to see whether the norm of the difference between this newly computed set of $I_{E,e}^{c_i}$ and the previously computed ones is smaller than ϵ_{thresh}. If yes, then terminate; otherwise, set $I_{A,e}^{v_j} = I_{E,e}^{c_i}$ if v_j and c_i are connected by e.

(b) *Variable node update.*

For all $e \in E_R$, compute $I_{E,e}^{v_i}$ using equation (4.4). Set $I_{A,e}^{c_j} = I_{E,e}^{v_i}$ if c_j and v_i are connected by e. Go to step 3(a).

4) *Compute $I_{E,e}^{c_i}$ for $e \in E_L$.*

For all $e \in E_L$, compute $I_{E,e}^{c_i}$ using equation (4.6). The average of these $I_{E,e}^{c_i}$ is denoted by I_E and $(I_{A,in}, I_E)$ is a point on the EXIT function.

The method can be adapted for computing EXIT functions over other channels by using appropriate update equations. Moreover, it can be used to compute the EXIT function of the structured part of other codes that have a succinct protograph representation such as IRA codes.

We demonstrate the effectiveness of our method by comparing EXIT functions computed by our method and by the Monte Carlo simulation based method for two cases: the structured part of E^2RC codes and that of IRA codes on BIAWGN channels respectively. The structured part of the E^2RC code has a protograph structure of size 128. All check nodes have degree 8. To get smooth curves, we apply 10^6 a priori inputs in Monte Carlo simulations for computing each point on the curves. As shown in Table 4.1, the maximum absolute error (MAE) between the EXIT functions computed using the two methods is less than 0.0072.

Table 4.1 Comparison of approaches for computing EXIT functions

AWGN: noise variance= 0.95775;10^4 (I_A, I_E) pairs generated				
	E^2RC		IRA	
Method	Proposed	Simulation	Proposed	Simulation
Computing time (s)	3.7	24596	0.6	175886
MAE	0.0072	-	0.00719	-

4.4.2 Code Design Examples

In our first example, we design a semi-structured E^2RC code with $d_{v,\max} = 7$ and check degree distribution of

$$\rho_6 = 0.339623, \rho_7 = 0.660377$$

Thus, from a complexity perspective these codes are comparable to the first design example in Section V in Song, etc. (2008). The mother code is of rate 0.5 and the H_2 part has a protograph structure of size $M = 32$. Our optimized code (referred to as code 0) is specified by

$$\lambda_3 = 0.4243, \lambda_7 = 0.5757$$

for $v \in V_1$ and has an asymptotic gap of 0.38 dB to capacity at rate 0.5. Fig. 4.3 gives the simulation results of this code of block length 2048 bits generated by the algorithms in Tian, etc. (2004), Ramamoorthy & Wesel (2004). For comparison, we also list the simulation results of the reference code and original E^2RC code from Song, etc. (2008). In this chapter, our codes follow the designed puncturing patterns of original E^2RC codes in Kim, etc. (2009) to get all puncturing code rates. From the simulation results it is clear that our code is better than the codes in Song, etc. (2008) for all code rates. In particular they do not suffer from the high error floor problem of original E^2RC codes at the mother code rate.

In our second example, we design another semi-structured E^2RC codes with concentrated check degree 8 and $d_{v,max} = 20$. The optimized code (referred to as code 1) is given by

$$\lambda_3 = 0.305825, \lambda_7 = 0.213474, \lambda_8 = 0.181737, \lambda_{20} = 0.298964$$

for $v \in V_1$ and it has an asymptotic gap of 0.217 dB to capacity at rate 0.5 which is smaller than code 0. This is expected since $d_{v,max}$ is higher in this case. The simulation results of code 1 of block length 16384 bits are given in Fig. 4.4. Also given are the simulation results of the E^2RC code that is constructed according to the degree distribution specified in Kim, etc. (2009) (referred to as original E^2RC). It shows that our code achieves slightly better performance at rates near mother code rate but suffers a little at higher code rates.

We use the following terminology in this chapter. The predicted threshold refers to the decoding threshold from asymptotic code performance analysis and the measured threshold refers to the channel parameter where the code achieves BER $= 10^{-4}$ in simulations. For our code 0 of length 2048 bits, the measured threshold at rate 0.5 is 1.47 dB which is 0.9 dB away from the predicted one. For our code 1 of length 16384 bits, this gap is only 0.4 dB.

4.4.3 Puncturing Performance Analysis and Joint Optimization of Semi-Structured E^2RC Codes

The puncturing performance of a given code is specified in terms of its decoding thresholds at all punctured code rates. The given semi-structured E^2RC codes are specified by $\lambda_{d_v}, v \in V_1$ and the knowledge of the protograph structure of the structured part. For a given channel parameter, we can compute the two EXIT functions using (4.3) and the approach of Section 4.4.1. Note that when computing EXIT functions at different puncturing code rates, we follow the designed puncturing pattern of E^2RC codes. The decoding threshold at a given code rate is determined by finding the channel parameter where the two EXIT curves (computed under the puncturing pattern at that rate) just begin to separate.

Our puncturing performance analysis of code 1 suggests that it has asymptotic decoding thresholds of around 0.40, 0.85, 1.40, 2.45, 3.44 dB at rates $\frac{8}{16}, \frac{8}{14}, \frac{8}{12}, \frac{8}{10}$ and $\frac{8}{9}$ respectively. The measured thresholds at these rates for the code of block length 16384 bits based on the simulation results in Fig. 4.4 are around $0.80, 1.22, 1.75, 2.78, 3.76$ dB, which are consistent with the predicted ones with gaps uniformly around 0.35 dB.

Joint Optimization of Semi-Structured E^2RC Codes

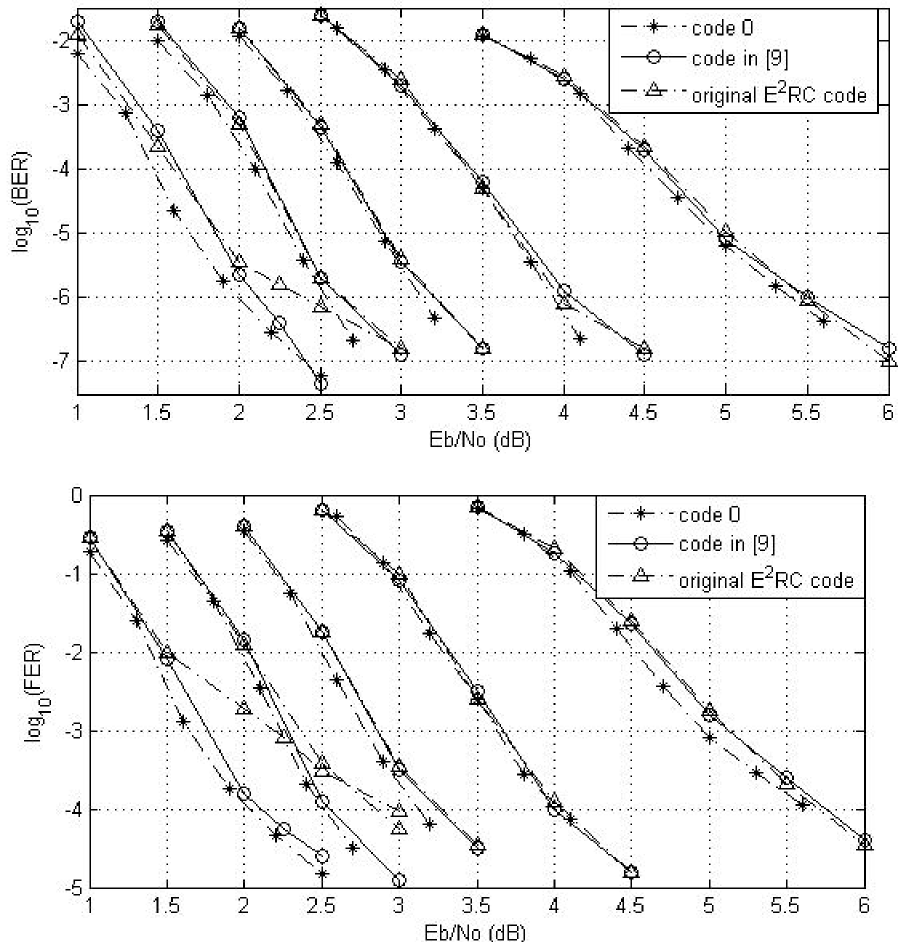


Figure 4.3 Comparison between our code 0 and the reference code in Song, etc. (2008) of block length 2048 bits. The code rates are 0.5, 0.6, 0.7, 0.8 and 0.9 from left to right. The figure on the top (bottom) corresponds to BER (FER).

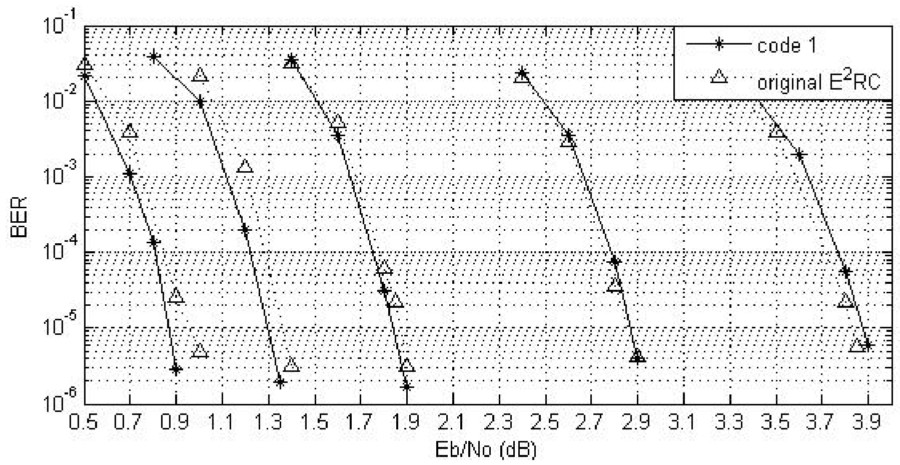


Figure 4.4 Comparison between our second code example and original E^2RC code in Kim, etc. (2009) of block length 16384 bits. The code rates are $0.5, 0.5714, 0.6667, 0.8$ and 0.8889 from left to right.

We now demonstrate that we can design our codes such that they have a small gap to capacity at all puncturing rates. This is because our puncturing pattern is deterministic and allows the determination of the asymptotic threshold at any puncturing rate for any given $\{\lambda_{d_v}, v \in V_1\}$. Let $\mathbf{R}$ be a specified set of code rates where we want to optimize the code. Let $\sigma(g, R_i)$ denote the channel noise parameter that is at a gap of g from the channel parameter corresponding to the Shannon limit at rate R_i. Let $I_{A,S}(I_{E,S}, \sigma(g, R_i))$ denote the plot of $I_{A,S}$ vs. $I_{E,S}$ under the puncturing pattern corresponding to rate R_i, at the channel parameter $\sigma(g, R_i)$. The notation $I_{A,unS}(I_{A,unS}, \sigma(g, R_i))$ will be used analogously. We can formulate the joint optimization problem as minimizing the maximum gap to capacity at all rates in $\mathbf{R}$ as follows.

Joint optimization algorithm

for $g = g_{min} : g_{max}$

```
    **Solve** the following linear program optimization problem
        maximize : $\sum_{d_v=1}^{d_{v,max}} \frac{\lambda_{d_v}}{d_v}$
        subject to : 1. $\sum_{d_v=1}^{d_{v,max}} \lambda_{d_v} = 1, \lambda_{d_v} \geq 0,$
                     2. $I_{E,unS}(I_{A,unS}, \sigma(g, R_i)) > I_{A,S}(I_{E,S}, \sigma(g, R_i))$
                        for $I_{A,unS} = I_{E,S} \in [0, 1)$, for all $R_i \in \mathbf{R}$
    **if** mother code rate corresponding to $\{\lambda_{d_v}\}$ is acceptable
        break; return $\{\lambda_{d_v}\}$ and $g$.
    **endif**
**endfor**
```

Note that the second set of constraints is the zero-error constraint by ensuring an iterative decoding tunnel for all EXIT charts at all the required code rates. We can obtain all the required EXIT functions relatively quickly using our approach outlined previously under the puncturing patterns for each R_i. To obtain optimized $\{\lambda_{d_v}\}$, we basically keep increasing g until we get the desired code rate. The code specified by λ_{d_v} is guaranteed to have asymptotic performance gap to capacity no larger than g at all code rates in $\mathbf{R}$.

We designed a semi-structured E^2RC code that was jointly optimized across the rate range $\frac{8}{16} \sim \frac{8}{9}$, where $M = 32$, all check nodes have degree 8 and $d_{v,\max} = 20$. The code is specified by

$$\lambda_3 = 0.309090, \lambda_6 = 0.278794, \lambda_{20} = 0.412116$$

Fig. 4.5 gives the simulation results for the code of block length 16384 bits (listed as code 2). Also plotted are the simulation results in Fig. 4.4 for code 1 and original E^2RC code from Section 4.4.2. The puncturing performance analysis suggests that code 1 has asymptotic performance gaps around $0.21, 0.32, 0.34, 0.41, 0.405$ dB to capacity at rates $\frac{8}{16}, \frac{8}{14}, \frac{8}{12}, \frac{8}{10}$ and $\frac{8}{9}$ respectively while code 2 has much more uniform gaps of around $0.29, 0.30, 0.25, 0.29, 0.295$ dB to capacity at these rates. The simulation results in Fig. 4.5 also suggest uniformly better performance of code 2 compared to code 1 across the range of rates. Moreover, the measured thresholds of the two codes at all rates in simulations are in good agreement with the predicted thresholds. At all code rates, the gap between the measured threshold and the predicted threshold is around 0.35 dB. Finally, we note that code 2 achieves better or at least the same performance as original E^2RC code at all rates.

The methods described in this section apply more generally to codes that have a structured component with a protograph representation, such as IRA codes. We applied our method to the design of IRA codes as well and obtained jointly optimized codes with performance gaps around $0.27, 0.30, 0.195, 0.27, 0.29$ dB to capacity at rates $\frac{8}{16}$,$\frac{8}{14}$, $\frac{8}{12}$, $\frac{8}{10}$ and $\frac{8}{9}$ respectively. Though not presented here, the simulation results of the IRA code are almost identical to the jointly optimized E^2RC code (code 2) discussed above.

To the best of our knowledge, a joint optimization algorithm that minimizes the gap to capacity simultaneously across all code rates has not been considered previously in the literature. In Ha, etc. (2004 - 1), the authors found the optimal puncturing patterns for two optimized mother codes (referred to as Ha code 1 and Ha code 2) and also gave their asymptotic puncturing performance using Gaussian approximation based density evolution. In Fig. 4.6, we compare the asymptotic thresholds of their codes with our code families. Note that our codes have the same maximum variable node degree and slightly smaller average variable node degree compared to the codes in Ha, etc. (2004 - 1). We observe that the gaps to capacity for our codes remain more or less the same across the range of rates, whereas the codes in Ha, etc. (2004 - 1) exhibit a larger gap to capacity at higher code rates.

4.5 Protograph E^2RC Codes Construction

In this section, we introduce the construction of a class of structured E^2RC-like codes based on protographs. The basic idea in these codes is to impose a protograph structure on the systematic part H_1 of the parity-check matrix as well (in addition to the protograph structure on H_2). We obtain a family of protographs with asymptotic gaps to capacity no larger than 0.28 dB across a wide range of rates when the maximum variable node degree is twenty. These codes have excellent finite length performance as well. In this part of the work,

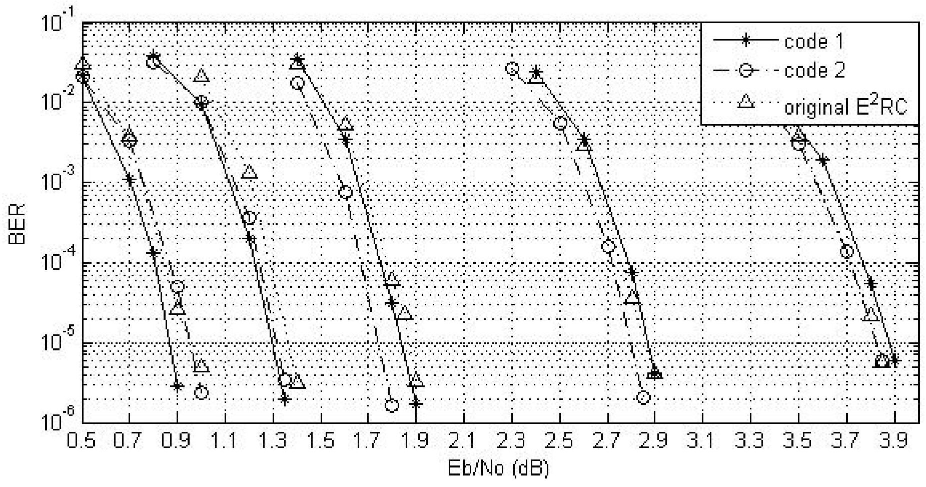


Figure 4.5 Comparison between two semi-structured E^2RC codes optimized at mother code rate (code 1) and simultaneously optimized at multiple rates (code 2) and original E^2RC code of block length 16384 bits. The code rates are $0.5, 0.5714, 0.6667, 0.8$ and 0.8889 from left to right.

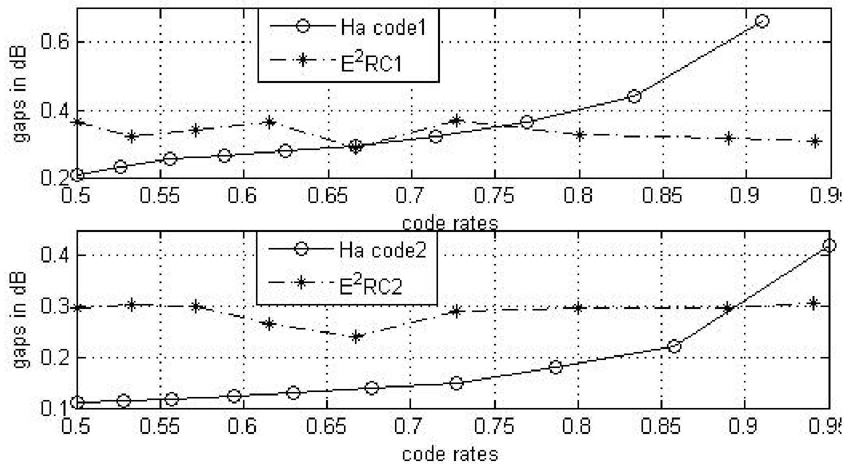


Figure 4.6 Asymptotic performance comparison between our codes and those from Ha, etc. (2004 - 1).

we use the reciprocal channel approximation of density evolution for computing the threshold for a given protograph Thorpe (2003), Chung (2000), Richardson (2009). The construction algorithm goes as follows.

1) *Find a good high-rate protograph.*

Using density evolution we first identify a high-rate protograph (starting protograph) with a low threshold.

2) *Use check-splitting to obtain lower-rate protographs with low thresholds.*

From the starting protograph, we perform check splitting in a systematic manner to obtain a family of good rate-compatible punctured protographs where the parity part has the E^2RC structure. These correspond to different code rates of the rate-compatible code family.

3) *Construct the LDPC code by replacing protograph edges with carefully chosen circulant permutations.*

With the protograph of mother code rate constructed from above steps, a larger graph defining the LDPC code is constructed by replacing the protograph edges with appropriately chosen circulant permutations by using techniques in Hu, etc. (2001), Tian, etc. (2004), Ramamoorthy & Wesel (2004), Weng, etc. (2004).

In the sequel we shall attempt to explain the construction process by means of an example. However, it should be clear that the techniques are applicable in general.

4.5.1 Starting Protograph

A protograph of size $M_0 \times N_0$ (M_0 - number of constraint nodes, N_0 - number of variable nodes) with low threshold serves as a starting point; we call it the starting protograph. Given the desirable code rate range

$$R_{min} \leq R \leq R_{max}$$

we decide the size of the mother code protograph $M \times N$ and the size of the starting protograph $M_0 \times N_0$, such that

$$\frac{N-M}{N} \leq R_{min}, \frac{N_0 - M_0}{N_0} \geq R_{max} \text{ and } N - M = N_0 - M_0$$

These conditions guarantee that the desirable code rate range is achievable by the construction. In addition, these parameters should be kept relatively small, say less than 50, to keep the construction complexity manageable. The construction will give a family of protographs with code rates ranging from $\frac{N-M}{N}$ to $\frac{N_0-M_0}{N_0}$. We impose the constraint that the degree of all variable nodes in the starting protograph is at least three. This is because the presence of degree two nodes causes a relatively high error floor.

In our example, $R_{min} = 0.5$, $R_{max} = 0.85$, and the size parameters are decided as

$$M_0 = 1, N_0 = 9 \text{ and } M = 8, N = 16$$

The protograph shown in Figure 4.7 is used as our starting protograph. It consists of one check node and nine variable nodes of degree {24,8,3,3,3,3,3,3,3} respectively. It has a threshold of 3.27 dB which is 0.24 dB away from the theoretical limit of 3.03 dB.

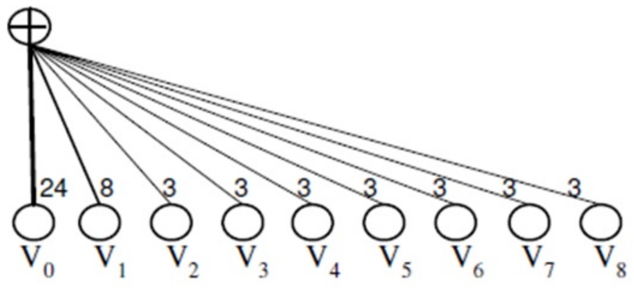


Figure 4.7 Example of a starting protograph. The number next to an edge denotes the number of parallel edges between the variable and the check node.

4.5.2 Check-Splitting

Consider a check node c of degree d in the protograph. The operation of check-splitting proceeds as follows. We split c into two check nodes c_1 and c_2, such that the degree of c_1 is d_1

and the degree of c_2 is d_2 and $d = d_1 + d_2$. Next we introduce a punctured variable node v' and introduce edges $c_1 - v'$ and $c_2 - v'$, so that the degree of v' is two. This is shown pictorially in Fig. 4.8.

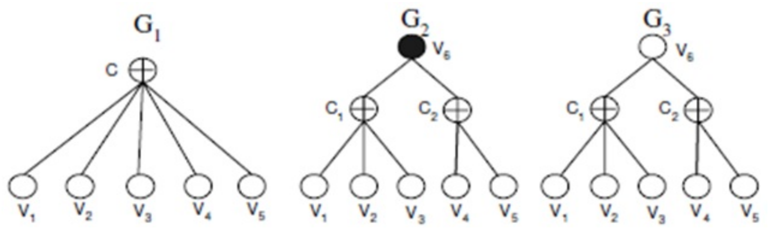


Figure 4.8 Check c with degree-5 is divided into c_1 and c_2. The new check nodes c_1 and c_2 are connected by a new degree-2 variable node v_6.

Check-splitting was used in Divsalar. etc. (2006) to construct protograph LDPC codes with linear minimum distance. A rigorous proof was given in Pishro-Nik & Fekri (2007) about the equivalence of the asymptotic decoding performance between a non-punctured high-rate LDPC code and a punctured low-rate LDPC code whose parity-check matrix can be built from that of the high-rate LDPC code through check-splitting. This is confirmed by our density evolution analysis on the first two graphs in Figure 4.8, which give the same asymptotic iterative decoding threshold.

In this work we use check-splitting as follows. We start with the high-rate protograph (as explained above) and split its check nodes in a specific manner. To obtain lower rates, after splitting a given check node, we convert the newly introduced punctured node into a transmitted node (e.g. see G_3 in Fig. 4.8). By applying check-splitting to a protograph of higher rate codes repeatedly, we finally arrive at the protograph of low rate mother code. In fact, the protographs produced in the check-splitting process form a family of rate-compatible protographs if we consider the newly added degree-two variable nodes in check-splitting as parity nodes that are used to provide incremental redundancy. However we note that the check-splitting needs to be done carefully (as discussed later), otherwise we may not have good code performance across all rates.

Table 4.2 Starting protograph

	old									new
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	
c_0	24	8	3	3	3	3	3	3	3	-

4.5.3 Constructing Protographs with E^2RC Like Structure

In this subsection we present a specific check-splitting technique that ensures the parity part of the code has the E^2RC structure. In what follows we call the original variable nodes of degree at least three in the starting protograph, *old nodes* and the variable nodes of degree two introduced due to check-splitting, *new nodes*.

During the construction, each check node has some connections with the old nodes and some connections with the new nodes. Accordingly, for a given check node we define its old node degree to be the number of connections to the old nodes and its new node degree to be the number of connections to the new nodes. Consider a check node c_0. When we split c_0 into c_{01} and c_{02}, a decision needs to be made on how the connections of c_0 are divided between them. To obtain the E^2RC structure, c_{01} is allocated all of c_0's new node degree. The parity node that is introduced in the check-splitting process has one connection to both c_{01} and c_{02} (refer to Theorem 22 for a complete proof). The old node degree also needs to be divided between c_{01} and c_{02} in a manner that ensures that the threshold of the new protograph is low. The division of the old node degree is discussed in more detail in the next subsection.

The construction proceeds in different stages. In each stage, we perform check-splitting on all check nodes in the current protograph. We now use the starting protograph in Figure 4.7 to demonstrate the process. Note that here we have $M_0 = 1$. We want to obtain a protograph with $M = 8$. Thus in this case we shall have $\log_2 8 = 3$ stages in the construction.

Let M_{n_s} denote the number of check nodes in the current protograph at the beginning of stage n_s. In this stage, we perform check-splitting on each of the M_{n_s} check nodes. Each check-splitting operation on the current protograph generates a protograph of next lower code rate. So at stage n_s, a set of M_{n_s} protographs of decreasing rates are generated. The order in

Table 4.3 The first splitting stage

	old									new
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
c_{01}	12	4	2	1	2	1	2	1	2	1
c_{02}	12	4	1	2	1	2	1	2	1	1

Table 4.4 The second splitting stage

	old									new	
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
c_{011}	6	2	1	1	1	0	1	1	1	1	1
c_{012}	6	2	1	0	1	1	1	0	1	0	1
c_{02}	12	4	1	2	1	2	1	2	1	1	0

which the check nodes are chosen for splitting can affect the thresholds at those rates.

We now show the first and second splitting stages for protograph in Figure 4.7. In the first stage, check-splitting on the single check node generates a new protograph of rate $\frac{8}{10}$.

In the second stage, there are two check nodes in the protograph for splitting. Density evolution analysis tells us that performing check-splitting on c_{01} first gives a protograph of code rate $\frac{8}{11}$ with a better decoding threshold than that of protograph generated by performing check-splitting on c_{02}. So in this stage, we first split c_{01} to generate a protograph of rate $\frac{8}{11}$ and then split c_{02} to generate a protograph of rate $\frac{8}{12}$. The corresponding protographs are shown below.

Table 4.5 The third splitting stage

	old									new		
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
c_{011}	6	2	1	1	1	0	1	1	1	1	1	0
c_{012}	6	2	1	0	1	1	1	0	1	0	1	0
c_{021}	6	2	1	1	0	1	1	1	0	1	0	1
c_{022}	6	2	0	1	1	1	0	1	1	0	0	1

The third stage proceeds in a similar manner. Refer to Fig. 4.9 for a graphical illustration of the construction process. Note that following this procedure we get node v_9 is 2-SR, while nodes v_{10} and v_{11} are 1-SR. More generally, we can show that when the algorithm finishes executing stage n_s, it adds M_{n_s} new parity nodes all of which are 1-SR. Consider parity nodes that existed at the beginning of stage n_s as k-SR nodes. We show in Theorem 22 that at the end of stage n_s, all those nodes become $(k+1)$-SR. It turns out that the construction ensures that half of the parity nodes in the final protograph are 1-SR, one-fourth are 2-SR and so on i.e. our construction technique results in protographs that have the E^2RC structure.

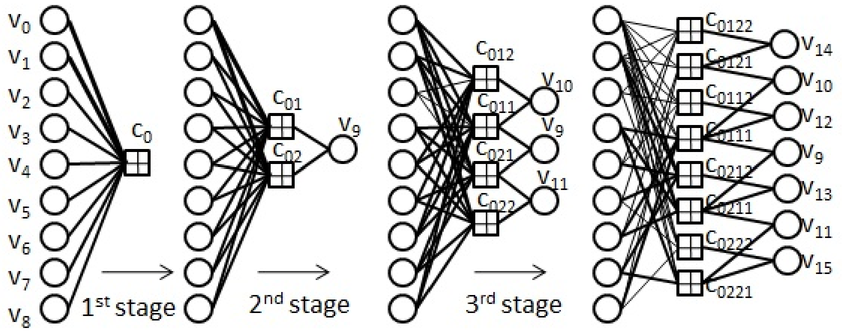


Figure 4.9 Protograph construction from check-splitting.

We puncture the parity nodes of the resultant protograph to obtain higher rates. The puncturing order is the inverse of the order in which the parity nodes are added during the construction i.e. a parity node that was added at the end will be punctured first, a parity node that was added at the penultimate stage will be punctured second and so on. This puncturing order ensures that the gap to capacity at the different rates is as predicted by the construction process.

4.5.4 Deciding Splitting Patterns

As pointed out before, in check-splitting, the old node degree of a given check node can be divided among the new check nodes in many ways. Now we explain how to decide proper splitting patterns to ensure good code performance. Let c_0 be the check node that needs to be split. Let s_0 be the vector of connections between c_0 and the old nodes e.g. $s_0 = [24\ 8\ 3\ 3\ 3\ 3\ 3\ 3\ 3]$. The splitting pattern refers to the set of vectors $s_{01} = [12\ 4\ 2\ 1\ 2\ 1\ 2\ 1\ 2]$ and $s_{02} = [12\ 4\ 1\ 2\ 1\ 2\ 1\ 2\ 1]$ that determine the connections between the nodes c_{01} and c_{02} and the old nodes. Thus $s_0 = s_{01} + s_{02}$.

Our search for good splitting patterns is guided by two main points.

a) *Trade-off between the performance of high-rate and low-rate protographs.*

In our experiments, we have found that there exists a tradeoff between the performance of high-rate and low-rate protographs during the construction. For example, it is possible to obtain very low thresholds for the higher rate protographs, however this typically comes at the expense of higher thresholds for the low-rate protographs in a later stage of the construction.

b) *Uniform splitting patterns give good performance.*

Note that considering all possible splitting patterns at all possible stages is essentially computationally infeasible since the number of possible splitting patterns grows exponentially. We have found that splitting patterns that split the connections roughly equally between the new check nodes have good performance across all code rates in the family. This reduces the search space a lot and it becomes possible to perform density evolution analysis (using the fast reciprocal channel approximation Chung (2000)) to determine the thresholds.

In the tables shown below we present two of the protographs that we have constructed using the construction algorithm described above. Both protographs are constructed from starting protograph in Figure 4.7. At the bottom of each table we show the gap to Shannon limit for the protograph at different puncturing levels for rates 8/9−8/16 from left to right. For protograph-1 , there is a good balance on the code performance at all code rates. For protograph-2, we chose the splitting patterns to lower the threshold for the high rate protographs, which resulted in somewhat worse performance at the low code rates.

Table 4.6 Mother code protograph-1

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
3	1	1	0	1	0	0	1	0	1	1	0	1	0	0	0
3	1	0	1	0	0	1	0	1	0	0	0	1	0	0	0
3	1	0	0	1	0	1	0	0	0	1	0	0	0	1	0
3	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0
3	1	0	1	0	0	1	0	0	1	0	1	0	1	0	0
3	1	1	0	0	1	0	1	0	0	0	0	0	1	0	0
3	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1
3	1	0	1	0	1	0	0	1	0	0	0	0	0	0	1
Gap to Shannon limit in dB (rates 8/9 - 8/16)															
0.24 0.25 0.22 0.21 0.24 0.26 0.27 0.26															

Table 4.7 Mother code protograph-2

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
2	0	1	1	0	1	0	1	1	1	1	0	1	0	0	0
4	2	0	1	0	1	0	1	1	0	0	0	1	0	0	0
3	0	0	0	0	0	2	0	0	0	1	0	0	1	0	0
3	2	1	0	2	0	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
3	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1
3	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0
3	4	1	0	0	0	0	0	1	0	0	0	0	0	1	0
Gap to Shannon limit in dB (rates 8/9 - 8/16)															
0.24 0.17 0.19 0.24 0.21 0.23 0.35 0.36															

4.5.5 Comparison with Previous Results

Here we give comparisons with some existing results in the literature. The asymptotic performance comparison between the protograph E^2RC code family represented by protograph-1 above and the AR4JA code family in Divsalar, etc. (2006) is given in Figure 4.10. Higher rate codes in the protograph E^2RC code family are obtained by puncturing parity nodes of the same type as v_{15}, v_{14},...v_9 in turn. We note that the thresholds of our protographs are closer to the Shannon limit than that of AR4JA codes for all the rates $1/2, 2/3$ and $4/5$. The average variable node degree of our codes is a little higher than that of AR4JA family. Note, however that the codes in Divsalar, etc. (2006) are not rate-compatible punctured codes.

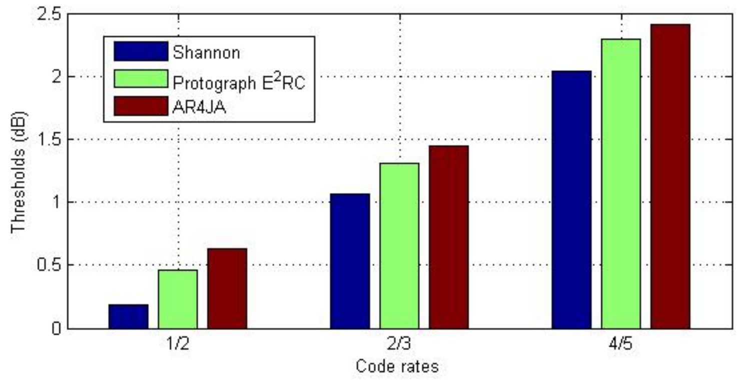


Figure 4.10 Asymptotic performance comparison between protograph E^2RC codes and AR4JA codes.

4.6 Conclusions

The E^2RC codes were proposed in Kim, etc. (2009) as a promising class of rate-compatible codes. In this work we introduced semi-structured E^2RC-like codes and protograph E^2RC codes. We developed EXIT chart based methods for the design of semi-structured E^2RC-like codes that allow us to determine near-optimal degree distributions for the systematic part of the code while taking into account the structure of the deterministic parity part. We presented a novel method for finding EXIT functions for structured code components that have a succinct protograph representation that is applicable in other scenarios as well. This allows us to analyze the puncturing performance of these codes and obtain codes that are better than the original construction. Using our approach we are able to jointly optimize the code performance across the range of rates for our rate-compatible punctured codes. Finally we consider E^2RC-like codes that have a protograph structure (called protograph E^2RC codes) and propose design rules for rate-compatible protographs with low thresholds. These codes are useful in applications since the protograph structure facilitates implementation. For both the semi-structured and protograph E^2RC families we obtain codes with small gaps to capacity across the range of rates.

Theorem 22 *Our construction generates E^2RC-like codes structure, that is, at the end of stage $k+1$ there will be $M_0 \cdot 2^k$ 1-SR nodes, $M_0 \cdot 2^{k-1}$ 2-SR nodes, $\ldots$, and M_0 (k+1)-SR nodes.*

Proof

We first recall the precise construction rule. Let c_0 be a check node with a certain number of connections to *new* nodes of degree-2. When c_0 is split into c_{01} and c_{02}, c_{01} inherits all of c_0's connections to new nodes and both c_{01} and c_{02} have one connection with the newly introduced variable node.

Suppose that the starting protograph is of size $M_0 \times N_0$. At the end of stage k of the construction algorithm, it is clear that there will be $M_0 \cdot 2^k$ check nodes and $M_0(2^k-1)$ new degree-2 parity nodes. Our aim is to show that this construction algorithm results in an H_2 part with the E^2RC structure. i.e. at the end of stage k of the algorithm, there are $M_0 \cdot 2^{k-1}$ 1-SR nodes, $M_0 \cdot 2^{k-2}$ 2-SR nodes, $\ldots$, and M_0 k-SR nodes. We proceed by induction.

Base Case. At the end of the first stage, we will have $2M_0$ check nodes and M_0 new degree-2, 1-SR parity nodes.

Inductive Step. Suppose that the statement is true at the end of stage k. We will show that it is true at the end of stage $k+1$. To see this note that at the end of stage $k+1$ we will have $M_0 \cdot 2^{k+1}$ check nodes formed by splitting the check nodes at the end of stage k. According to the construction algorithm, $M_0 \cdot 2^k$ check nodes will inherit the previous connections while the remaining will have just one connection to the $M_0 \cdot 2^k$ newly introduced degree-2 variable nodes. Therefore we will have at least $M_0 \cdot 2^k$ 1-SR nodes at the end of stage $k+1$. Next we note that any node that was of type α-SR at the end of stage k will now become of type (α+1)-SR. This is because each of the check nodes it is connected to will have one additional connection. This implies that at the end of stage $k+1$ there will be $M_0 \cdot 2^k$ 1-SR nodes, $M_0 \cdot 2^{k-1}$ 2-SR nodes, ..., and M_0 (k+1)-SR nodes. This shows the required result.

CHAPTER 5. SUMMARY AND DISCUSSION

This thesis describes an improved combinatorial algorithm for computing the unicast capacity of deterministic wireless networks and considers design and analysis of rate-compatible LDPC codes that have potential applications in communication systems with varying channel conditions, like the channels in the wireless networks that is considered above. The first part of the work contributes to efficiently computing the capacity of deterministic wireless networks which in turn provides insight into the study of capacity and transmission schemes in the original wireless networks. The second part of the work considers design and analysis of efficient and practical channel codes that are expected to have uniformly good performance under varying channel conditions in wireless networks.

Characterizing the capacity of wireless networks has been a challenging problem for decades. The deterministic channel model for wireless networks, the ADT model by Avestimehr, etc. (2007 - 1), has been a promising approach to study the wireless information flow. The unicast and multicast capacities of deterministic wireless networks were characterized by the authors in the original paper for the model Avestimehr, etc. (2007 - 1). However efficient algorithms are not implied by the characterization. Several works have been proposed on solving the problem efficiently. The first work on computing unicast capacity of deterministic wireless networks was by Amaudruz & Fragouli (2009). Our work improves upon the original algorithm by Amaudruz & Fragouli for computing the unicast capacity of linear deterministic wireless networks. We amend the original algorithm so that it finds the unicast capacity correctly for any given deterministic networks. Moreover we fully explore several useful combinatorial features intrinsic in the problem which lead to reduced complexity. Our improved algorithm applies with any size of finite field associated with the ADT model defining the network. Our improved algorithm proves to be very competitive when comparing with other algorithms, Yazdi & Savari (2009), Goemans, etc. (2009), on solving the same problem in terms of complexity.

After identifying the capacity and corresponding transmission scheme for a wireless network, efficient and practical channel codes are desirable to fulfill the goal of approaching the theoretical limit or capacity established. In the second part, our work on the design and analysis of E^2RC codes aims to fulfill this goal. Moreover, the previous work on the design and analysis of LDPC codes are all targeting at a specific code rate and no work is known on the design and analysis of rate-compatible LDPC codes so that the code performance at all code rates in the family is manageable and predictable. In our work, we proposed algorithms for the design and analysis of rate-compatible LDPC codes with good puncturing performance and make the code performance at all code rates manageable and predictable. Our work is based on E^2RC codes, while our approaches in the design and analysis can be applied more generally not only to E^2RC codes, but to other suitable scenarios, like the design of IRA codes (Jin, etc. (2000)). Most encouragingly, we obtain families of rate-compatible codes whose gaps to capacity are at most 0.3 dB across the range of rates when the maximum variable node degree is twenty, which is very promising compared with other existing results.

Efficiently-encodable rate-compatible LDPC codes are expected to have good error correction performance under varying channels conditions like in wireless networks. E^2RC codes were proposed as a promising class of rate compatible codes. We first presented systematic design and analysis of semi-structured E^2RC codes using EXIT chart. We proposed a new analytical method of computing the EXIT functions of structured parts of E^2RC codes without resorting to Monte Carlo simulations. The design of capacity-approaching E^2RC code was accomplished by linear programming. By exploring the E^2RC structure and its designed puncturing pattern, we present the method for puncturing performance analysis and propose the joint optimization algorithm for designing rate-compatible punctured codes that are simultaneously optimized at any specified set of code rates. Our jointly optimized codes with $d_{v,max} = 20$ have performance gaps to capacity no larger than 0.3 dB across the entire rate range.

We then proposed a construction of rate-compatible codes based on protographs inspired by the E^2RC codes. Protograph E^2RC codes have protograph representations which facilitate their asymptotic performance analysis and allow the implementation of high speed decoders. The construction starts with a high rate protograph with low threshold. Protographs of lower rate codes are iteratively derived from the higher rate protographs via the process of check-splitting. The check-splitting process is specially designed to ensure that the parity nodes in the protograph have the E^2RC structure. Furthermore the construction process guided by density evolution also produces protographs that have low thresholds at all rates. Thus, it introduces a systematic technique for the design of E^2RC-like codes. Both asymptotic performance analysis and simulation results demonstrate that the protograph E^2RC code family has good performance across all code rates.

APPENDIX
IMPLEMENTATION OF DENSITY EVOLUTION FOR PROTOGRAPHS

Based on the description of Appendix B in Richardson & Urbanke (2008), the author implements the density evolution for protographs and uses it in the work in this thesis. This chapter gives the implementation Matlab codes that are used throughout this thesis.

.1 Instructions

This is a brief introduction to the free software of density evolution functions for computing the asymptotic decoding threshold of a given protograph LDPC code based on its protograph representation.

.1.1 Software Package

- DEFuncSCZ2010.m, the main file with the calling function DEFuncSCZ2010,
- cap_func.mexw64, helper function for computing capacity based on a given channel parameter and
- cap_inv_func.mexw64, another helper function for deriving the threshold of channel parameter giving a capacity/rate value

.1.2 Software Usage

Put cap_func.mexw64 and cap_inv_func.mexw64 in the same folder as DEFuncSCZ2010 function. Call DEFuncSCZ2010(filename) for running the algorithm.

.1.3 Input

Input filename is a string of filename containing the protograph description of the given protograph LDPC codes.

Format for the protograph representation: it's the adjacency matrix description of the corresponding Tanner graph where rows correspond to check nodes and columns correspond to variable nodes.

.1.3.1 Example

{24 8 3 3 3 3 3 3 3} represents a protograph of one check node and nine variable nodes and the number of connections between them are {24,8,3,3,3,3,3,3,3} respectively. The protograph is shown in Fig. A.1.

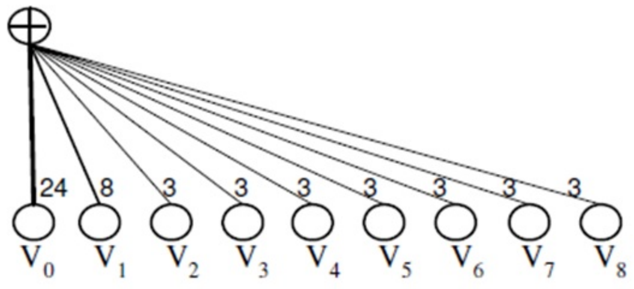


Figure A.1 Protograph example

.1.4 Outputs

it is under the assumption of binary input AWGN channel.

- the code rate of the given protograph LDPC codes
- the asymptotic decoding threshold computed from density evolution
- the Shannon theoretical decoding threshold for this code rate
- the gap in dB between the asymptotic decoding threshold of the given LDPC codes and the theoretical decoding threshold

.2 Matlab Implementation of Density Evolution

```
%------------------------------------------------------------------------
% Free software: Density evolution functions for computing the asymptotic decoding threshold
%                of a given protograph LDPC code based on its protograph representation
% Copyright:     Cuizhu Shi, Iowa State University
%                These codes may be used freely for academic, or non-commercial purposes
% Date:          Generated 03/01/2008, last revision: 02/15/2010
%------------------------------------------------------------------------

function DEFuncSCZ2010(filename)

iter=2000;
horg=textread(filename);
sizem=size(horg);
MM=sizem(1,1);
NN=sizem(1,2);
h=zeros(1,MM*NN);
for i=1:1:MM
    for j=1:1:NN
        h(1,(i-1)*NN+j)=horg(i,j);
    end
end

lowthres=0;
highthres=10;
stepini=1;
```

```
rate=(NN-MM)/(NN);
if (rate>1)&&(rate<0)
    fprintf(”error rate”);
end

results=cap_inv_func(rate);
del_ther=1/sqrt(results);

stope=10^-3;
stop0=0;

L=sum(h≠ 0);
delta=0;
thresh=0;

edgenum=zeros(1,MM*NN);
edgepos=zeros(1,L);
k=0;
for i=1:1:MM*NN
    if h(1,i)≠ 0
        k=k+1;
        edgenum(1,i)=k;
        edgepos(1,k)=i;
    end
end

for stepii=stepini:1:4
    intstep=10^-stepii;
```

```
% run DE for each channel parameter
for kkkk=1:1:floor((highthres-lowthres)/intstep)+1
    delta=lowthres+(kkkk)*intstep;
    if delta>del_ther-10^{-10}
        thresh=delta-intstep;
        lowthres=thresh;
        break;
    end
    m0=1/(delta^2);
    mu=zeros(1,L);
    mv=m0*ones(1,L);
    for iteration=1:1:iter
        stop0=1;
        mu=zeros(1,L);
        for i=1:1:L
            rowip=edgepos(1,i);
            for j=1:1:NN
                specp=(ceil(rowip/NN)-1)*NN+j;
                if h(1,specp)≠ 0
                    if (specp)≠rowip
                        mu(1,i)=mu(1,i)+h(1,specp)*cap_inv_func(1-cap_func(mv(1,edgenum(1,specp))));
                    end
                    if (specp)==rowip
                        if h(1,specp)>1
                            mu(1,i)=mu(1,i)+(h(1,specp)-1)*cap_inv_func(1-cap_func(mv(1,edgenum(1,specp))));
                        end
                    end
                end
```

```
  end
  if mu(1,i)>stope
    stop0=0;
  end
end
if stop0==1
  break;
end

mv=m0*ones(1,L);
for i=1:1:L
  rowip=edgepos(1,i);
  for j=1:1:MM
    tmpnumber=mod(rowip,NN);
    if tmpnumber==0
      tmpnumber=NN;
    end
    specp=(j-1)*NN+tmpnumber;
    if h(specp)≠0
      if (specp)≠rowip
        mv(1,i)=mv(1,i)+h(specp)*cap_inv_func(1-cap_func(mu(1,edgenum(1,specp))));
      end
      if (specp)==rowip
        mv(1,i)=mv(1,i)+(h(specp)-1)*cap_inv_func(1-cap_func(mu(1,edgenum(1,specp))));
      end
    end
  end
end
```

```
        end
        if stop0==0
            thresh=delta-intstep;
            lowthres=thresh;
            break;
        end
    end % end of each delta
end
dB=10*log10(1/(2*rate*thresh^2));

shann=10*log10(1/(2*rate)*results);
str=sprintf('DEFunc: rate=%.3f,threshold=%f/%f dB,Shann=%f/%f dB,gap=%f dB',
                        rate,thresh,dB,del_ther,shann,dB-shann);
disp(str);
res=dB;
```

If we run the DEFuncSCZ2010 function on the given example in Fig. A.1, we will get its decoding threshold 3.27dB. Another example is from Chapter 4, that is a protograph of {20,8,3,3,3,3,3,3,3} which has a decoding threshold 3.27dB (note here we rounded the threshold value to the second digit. This function was used in computing the decoding thresholds for all protographs in our work in Chapter 4.

BIBLIOGRAPHY

Tom Richardson, Ruediger Urbanke (2008). Modern Coding Theory. *Cambridge University Press*, May 2008

Amir Salman Avestimehr and Suhas N. Diggavi and David N C. Tse (2007). A Deterministic Approach to Wireless Relay Networks. *Proceedings of Allerton Conference on Communication, Control and Computing, Illinois*, Sep. 2007

Amir Salman Avestimehr and Suhas N. Diggavi and David N C. Tse (2007). Wireless Network Information Flow. *Proceedings of Allerton Conference on Communication, Control and Computing, Illinois*, Sep. 2007.

C. E. Shannon (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, volume 27, pages 379-423, 623-656, Jul., Oct. 1948

Thomas M. Cover and Joy A. Thomas (2006). Elements of Information Theory. *A John Wiley & Sons, Inc.*, 1991

Amir Salman Avestimehr and Suhas N. Diggavi and David N C. Tse (2009). Wireless Network Information Flow: A Deterministic Approach. *http://arxiv.org/abs/0906.5394*, 2009

Ahlswede, R. and Ning Cai and Li, S.-Y.R. and Yeung, R.W. (2000). Network information flow. *Information Theory, IEEE Transactions on*, vol. 46, pp. 1204C1216, 2000

Aurore Amaudruz and Christina Fragouli (2009). Combinatorial Algorithms for Wireless Information Flow. *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 555 - 564, Jan. 2009

S. M. Sadegh Tabatabaei Yazdi and Serap A. Savari (2009). A Combinatorial Study of Linear Deterministic Relay Networks. *Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing*, 2009

Michel X. Goemans and Satoru Iwata and Rico Zenklusen (2009). An Algorithmic Framework for Wireless Information Flow. *Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing*, 2009

Javad Ebrahimi and Christina Fragouli (2009). Combinatiorial Algorithms for Wireless Information Flow. *http://arxiv.org/abs/0909.4808*, Sep. 2009

R. Motwani and P. Raghavan (1995). Randomized Algorithms. *Cambridge University Press*, 1995

Claude Berge (1957). Two Theorems in Graph Theory. *Proceedings of the National Academy of Sciences*, vol. 43, no. 9, pages 842 - 844, Sep. 1957

Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein (2001). Introduction to Algorithms. *MIT press, Cambridge, MA*, Second Edition. 2001

Robert G. Gallager (1963). Low Density Parity Check Codes. *MIT press, Cambridge, MA*, 1963

M. Tanner (1981). A Recursive Approach to Low Complexity Codes. *IEEE Trans. Inform. Theory*, vol. IT-27, no. 5, pages 533 - 547, Sep. 1981

Jeongseok Ha and Jaehong Kim and Steven W. McLaughlin (2004). rate-Compatible Puncturing of Low-Density Parity-Check Codes. *IEEE Trans. Inform Theory*, vol. 50, no. 11, pages 2824 - 2836, Nov. 2004

Jeongseok Ha and Jaehong Kim and Steven W. McLaughlin (2004). Puncturing for Finite Length Low-Density Parity-Check Codes. *IEEE Int. Symp. Inform. Theory, Chicago*, pages 152, Jun. 2004

Jeongseok Ha and Jaehong Kim and Demijan Klinc and Steven W. McLaughlin (2006). Rate-Compatible Punctured Low-Density Parity-Check Codes with Short Block Lengths. *IEEE Trans. Inform. Theory*, vol. 52, no. 2, pages 728 - 738, Feb. 2006

Guosen Yue and Xiaodong Wang and Mohammad Madihian (2007). Design of Rate-Compatible Irregular Repeat Accumulate Codes. *IEEE Trans. Commun.*, vol. 55, no. 6, pages 1153 - 1163, Jun. 2007

Mohammad R. Yazdani and Amir H. Banihashemi (2004). On Construction of Rate-Compatible Low-Density Parity-Check Codes. *IEEE Comm. Letters*, vol. 8, no. 3, pages 159 - 161, Mar. 2004

Jaehong Kim and Woonhaing Hur and Ramamoorthy, A. and McLaughlin, S.W. (2006). Design of Rate-Compatible Irregular LDPC Codes for Incremental Redundancy Hybrid ARQ Systems. *IEEE Int. Symp. Inform. Theory*, pages 1139 - 1143, Jul. 2006

Jaehong Kim and Aditya Ramamoorthy and Steven W. McLaughlin (2009). Design of Efficiently-Encodable Rate-Compatible Irregular LDPC Codes. *IEEE Trans. on Communications*, vol. 57, no. 2, pages 365 - 375, Feb. 2009

J. Thorpe (2003). Low Density Parity Check (LDPC) Codes Constructed from Protographs. *JPL INP Progress Report 42-154*, Aug. 2003

Seungmoon Song and Daesung Hwang and Sunglock Seo and Jeongseok Ha (2008). Linear-Time Encodable Rate-Compatible Punctured LDPC Codes with Low Error Floors. *IEEE VTC*, pages 749 - 753, May. 2008

Hui Jin and Aamod Khandekar and Robert McEliece (2000). Irregular Repeat-Accumulate Codes. *Proc. 2nd Int. Symp. Turbo Codes and Related Topics, Brest, France*, pages 1 - 8, Sep. 2000

Aline Roumy and Souad Duemghar and Giuseppe Caire and Sergio Verdu (2004). Design Methods for Irregular Repeat-Accumulate Codes. *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pages 1711 - 1727, Aug. 2004

T. Richardson (2009). Multi-Edge Type LDPC Codes. *presented at the Workshop honoring Prof. Bob McEliece on his 60th birthday, California Institute of Technology, Pasadena, California*, May. 2002

Thomas J. Richardson and M. Amin Shokrollahi and Rdiger L. Urbanke (2001). Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes. *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pages 619 - 637, Feb. 2001

Thomas J. Richardson and Rdiger L. Urbanke (2001). The Capacity of Low-Density Parity Check Codes Under Message-Passing Decoding. *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pages 599 - 618, Feb. 2001

J. Hagenauer (1988). Rate compatible punctured convolutional codes (rcpc codes) and their applications. *IEEE Trans. Commun.*, vol. 36, no. 4, Apr. 1988

Michael G. Luby and Michael Mitzenmacher and M. Amin Shokrollahi and Daniel A. Spielman and Volker Stemann (1997). Practical Loss-Resilient Codes. *Proc. 29th Annu. ACM Symp. Theory of Computing*, pages 150 - 159, 1997

M. Amin Shokrollahi (1999). New Sequences of Linear Time Erasure Codes approaching the Channel Capacity. *Proc. 13th International Symposium Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 65 - 76, 1999

Stephan ten Brink and Gerhard Kramer (2003). Design of Repeat-Accumulate Codes for Iterative Detection and Decoding. *IEEE Trans. Signal Processing*, vol. 51, no. 11, pages 2764 - 2772, Nov. 2003

Sae-Young Chung (2000). On the Construction of Some Capacity-Approaching Coding Schemes. *Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge,Massachusetts*, Sep. 2000

Stephan ten Brink (2001). Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes. *IEEE Trans. Commun.*, vol. 49, no. 10, pages 1727 - 1737, Oct. 2001

Stephan ten Brink and Gerhard Kramer and Alexei Ashikhmin (2004). Design of Low-Density Parity-Check Codes for Modulation and Detection. *IEEE Trans. Commun.*, vol. 52, no. 4, pages 670 - 678, Apr. 2004

Alexei Ashikhmin and Gerhard Kramer and Stephan ten Brink (2004). Extrinsic Information Transfer Functions: Model and Erasure Channel Properties. *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pages 2657 - 2673, Nov. 2004

Eran Sharon and Alexei Ashikhmin and Simon Litsyn (2006). Analysis of Low-Density Parity-Check Codes Based on EXIT Functions. *IEEE Trans. Communications*, vol. 54, no. 8, pages 1407 - 1414, Aug. 2006

Tao Tian and Christopher R. Jones and John D. Villasenor and Richard D. Wesel (2004). Selective Avoidance of Cycles in Irregular LDPC Code Construction. *IEEE Trans. Commun.*, vol. 52, no. 8, pages 1242 - 1247, Aug. 2004

Aditya Ramamoorthy and Richard Wesel (2004). Construction of Short Block Length Irregular Low-Density Parity-Check Codes. *IEEE International Conference on Communications*, vol. 1, pages 410 - 414, 2004

Xiao-Yu Hu and Evangelos Eleftheriou and Dieter-Michael Arnold (2001). Progressive Edge-Growth Tanner Graphs. *IEEE GlobeCom*, vol. 2, pages 995 - 1001, Nov. 2001

Wen-Yen Weng and Ramamoorthy, A. and Wesel, R.D. (2004). Lowering the error floors of irregular high-rate LDPC codes by graph conditioning. *IEEE Vehicular Technology Conference*, vol. 4, pages 2549 - 2553, Sep. 2004

Dariush Divsalar and Sam Dolinar and Christopher Jones (2006). Construction of Protograph LDPC Codes with Linear Minimum Distance. *IEEE Int. Symp. Inform. Theory*, pages 664 - 668, Jul. 2006

Hossein Pishro-Nik and Faramarz Fekri (2007). Results on Punctured Low-Density Parity-Check Codes and Improved Iterative Decoding Techniques. *IEEE Trans. Inform. Theory*, vol. 53, no. 2, pages 599 - 614, Feb. 2007

Dariush Divsalar and Sam Dolinar and Christopher Jones (2006). Construction of Protograph LDPC Codes with Linear Minimum Distance. *IEEE Int. Symp. Inform. Theory*, pages 664 - 668, Jul. 2006